INDEX

A

absorbing nodes, 217
absorption probability, 217
absorption time, 217
acyclic graphs, 136
adjacency list representation, 7–15
adjacency matrix representation, 15–17
admissible heuristics, 113. *See also* heuristics
Algorithmic Thinking (Zingaro), 369
Algorithms (Sedgewick and Wayne), 200
all-pairs shortest-path problem, 102–103. *See also* shortest-path problem
anti-parallel edges, 231, 253–254
arcs. *See* edges
articulation point finding algorithm, 186–191
    `articulation_point_dfs()` function, 188
    `articulation_point_root()` function, 188
articulation points, 177
Artificial Intelligence (Russell and Norvig), 323
A* search, 119–132
    admissible heuristics, 124
    `astar_dynamic()` function, 128
    `astar_search()` function, 120
    exploring graphs, 126–132
    puzzles, 124–126
augmenting paths, 238–243. *See also* maximum-flow problem
average clustering coefficient, 25

B

backward residual, 235, 239, 241
Bellman-Ford algorithm, 97–99
    `BellmanFord()` function, 99
    cost, 99
best-first search, 111, 114. *See also* heuristic-guided search
bipartite graphs, 255, 257–273
    `bipartite_labeling()` function, 259
    labeling, 258–263
bipartite matching problem, 263–273
    `bipartite_matching_exh()` function, 266
    `bipartite_matching_max_flow()` function, 270
    exhaustive algorithms, 264–269
    `matching_recursive()` function, 266
    using maximum-flow algorithms, 269–273
breadth-first search, 63–74
    `breadth_first_search()` function, 65
    maximum-flow problem, 246–251
    shortest paths, 68–70
bridge-finding algorithm, 180–186
    `bridge_finding_dfs()` function, 184
    cost, 184
    `find_bridges()` function, 184
bridges, 176

C

capacity, 230–232. *See also* maximum-flow problem
`CapacityEdge` class, 233–235
    `adjusted_used()` method, 234
    `capacity_left()` method, 234
    constructor, 234
    `flow_used()` method, 234
Chaitin, George, 293
cliques, 301–308
    backtracking search, 305–308
    `clique_expansion_options()` function, 303
    `clique_greedy()` function, 304
    greedy search, 303–305
    `is_clique()` function, 302

cliques *(continued)*
maximum clique, 302
maximum_clique_backtracking() function, 306
maximum_clique_recursive() function, 305
closed-neighborhood subgraphs, 27
clustering coefficient, 24–26
ave_clustering_coefficient() function, 25
clustering_coefficient() function, 24
comma-separated values (CSV), 347
connected components, 42, 176, 193. *See also* strongly connected components
dfs_recursive_cc() function, 55
and Eulerian cycles, 336
constructing graphs
build_graph_from_points() function, 354
from edge list, 346–347
loading co-occurrence graphs, 351–353
loading edges from file, 347–349
make_graph_from_dependencies() function, 355
make_graph_from_edges() function, 346
make_graph_from_multi_csv() function, 353
make_graph_from_weighted_csv() function, 348
make_graph_from_weighted_csv2() function, 351
from preconditions, 355–356
from spatial data, 353–355
co-occurrence graphs, 351–353
cut vertex, 177
cycles, 136
detecting, 147–148
negative, 96

D

DAG (directed acyclic graph), 136
Data Structures and Algorithms (Aho, Hopcroft, and Ullman), 199, 200
Data Structures the Fun Way (Kubica), xxi, 358
degree, 22–24, 335
computing, 336
in-degree, 23
out-degree, 23
and self-loops, 23, 336
depth-first search, 45–61
depth_first_search_basic_all() function, 49
depth_first_search_path() function, 50
depth_first_search_stack() function, 52
dfs_recursive_basic() function, 48
dfs_recursive_cc() function, 55
dfs_recursive_path() function, 49
recursive algorithm, 48–52
stack-based algorithm, 52–55
uses
checking reachability, 48
finding articulation points, 186–191
finding bridges, 180–186
finding connected components, 55–57
finding Hamiltonian paths, 328–330
Kosaraju-Sharir's algorithm, 199–204
maximum-flow problem, 241–243
solving the traveling salesperson problem, 331–334
topological sort, 143–147
depth-first search forest, 58
depth-first search tree, 57–59. *See also* DFSTreeStats class
back edge, 180
post-order index, 199
preorder index, 180
untraversed edges, 180–181
uses
finding articulation points, 186
finding bridges, 180
DFSTreeStats class, 182
constructor, 183

uses
finding articulation points, 188
finding bridges, 183
diameter, 108–110
GraphDiameter() function, 109
Dijkstra, Edsger W., 91
Dijkstra's algorithm, 91–96
cost, 93
Dijkstras() function, 92
disconnected graphs, 95–96
nonnegative edges, 91
Dinic, E. A., 246
Dinitz, Yefim, 246
Dinitz algorithm, 246–251
directed acyclic graph (DAG), 136
directed edges, 5–7. *See also* edges
directed graphs, 6
disjoint sets, 161, 370
distance between nodes, 90

E

|*E*|, 4
Edge class, 9–10
constructor, 9
representing undirected edges, 10
representing unweighted edges, 9
edges, 4
destination, 6
directed, 5–7
graphical representation, 4–6
to indicate order, 136
negative weight, 96–97
origin, 6
relaxation, 97
undirected, 6
weighted, 5
Edmonds, Jack, 246
Edmonds-Karp algorithm, 246–251. *See also* maximum-flow problem
augmenting path, 238
in bipartite graph matching, 270
edmonds_karp() function, 248
find_augmenting_path_bfs() function, 246
Euclidean distance, 112–113, 354
euclidean_dist() function, 112
Euler, Leonhard, 334, 335
Eulerian cycles, 334–341
finding with Hierholzer's algorithm, 337
has_eulerian_cycle() function, 336
is_eulerian_cycle() function, 336
Eulerian paths, 334
Eulerian trails, 335
expected absorption time, 218
expected hitting time, 218

F

finishing time, 199
flow, 230–231. *See also* maximum-flow problem
Floyd-Warshall algorithm, 103–108
cost, 105
dynamic programming, 104
FloydWarshall() function, 105
intermediate paths, 103
Ford, L.R., Jr., 237
Ford-Fulkerson algorithm, 237–246. *See also* maximum-flow problem
augmenting_path_dfs_recursive() function, 241
find_augmenting_path_dfs() function, 241
ford-fulkerson() function, 244
forward residual, 235, 239, 241
Fulkerson, D. R., 237

G

games of chance, 214–215, 219–221
Graph class, 12–15. *See also* constructing graphs
constructor, 12, 350
copying, 15
get_edge() method, 12
get_index_by_name() method, 350
get_in_neighbors() method, 21
insert_edge() method, 13
inserting nodes by name, 350–351
insert_node() method, 14
is_edge() method, 13
make_copy() method, 15
make_edge_list() function, 13, 100
make_undirected_neighborhood_subgraph() method, 27

Graph class *(continued)*
- numeric node index, 9, 12
- remove_edge() method, 13
- representing undirected graphs, 12, 14
- save_graph_to_csv() function, 350
- saving to file, 349–350

graph coloring, 277–297
- backtracking search, 284–290
  - graph_color_dfs() function, 285
  - graph_color_dfs_pruning() function, 287
  - pruning, 286–290
- exhaustive search, 283–284
  - cost, 283
  - graph_color_brute_force() function, 283
- first_unused_color() function, 290
- greedy search, 290–293
  - graph_color_greedy() function, 291
- is_graph_coloring_valid() function, 278
- minimum graph-coloring problem, 278
- node removal, 293–297
  - graph_color_removal() function, 293
- number of states, 282
- problem definition, 278

GraphMatrix class, 16–17, 212
- constructor, 16
- get_edge() method, 16
- in random walks, 212
- representing undirected graphs, 16
- set_edge() method, 17
- simulate_random_step() method, 212

graphs. *See also* constructing graphs; edges; nodes
- adjacency list representation, 7–15
- adjacency matrix representation, 15–17
- bipartite, 255, 257–273
- diameter, 108–110
- directed, 6
- transpose, 200–201
- undirected, 6
- unweighted, 5
- weighted, 5

graph search, 45

greedy best-first search, 114–118
- greedy_search() function, 115

grid-based graphs, 70–72
- height, 70
- indexing, 71
- make_grid_graph() function, 71
- make_grid_with_obstacles() function, 72
- and mazes, 72
- obstacles, 71–72
- width, 70

H

Hamilton, William, 326

Hamiltonian cycles, 326

Hamiltonian paths, 326–330
- finding with depth-first search, 328–330
- hamiltonian_dfs() function, 329
- hamiltonian_dfs_rec() function, 328
- is_hamiltonian_path() function, 327

heaps, 358–361
- as an array, 59
- dequeue, 366
- enqueue, 365
- HeapItem class, 358
- max heap, 358
- min heap, 358
- swapping elements, 359–361

heuristic-guided search, 111–132. *See also* A* search; greedy best-first search

heuristics, 112–114
- admissible, 113
- Euclidean distance, 112

hidden Markov model, 210

hierarchical clustering, 169

Hierholzer, Carl, 337

Hierholzer's algorithm, 337, 341
- hierholzers() function, 338
- subloop, 337

hitting probability, 217
hitting time, 217

I

in-degree, 23
independent set, 308–315, 321–322
  build_independent_set_random() function, 322
  heuristics, 310
  independent_set_expansion_options() function, 309
  independent_set_greedy() function, 311
  independent_set_lowest_expansion() function, 310
  independent_set_random() function, 321
  is_independent_set() function, 308
  maximum, 309
  maximum_independent_set_backtracking() function, 313
  maximum_independent_set_rec() function, 312
  using backtracking search, 311–312
  using greedy search, 309–311
in-neighbors, 21
Introduction to Algorithms (Cormen, Leiserson, Rivest, and Stein), 323
iterative deepening, 59–61

J

Jarník, Vojtěch, 156

K

Kahn, Arthur B., 139
Kahn's algorithm, 139, 143
  check_cycle_kahns() function, 148
  cost, 141
  Kahns() function, 140
Karp, Richard M., 246
Kosaraju, S. Rao, 199
Kosaraju-Sharir's algorithm, 199–204
  add_reachable() function, 201
  cost, 200
  kosaraju_sharir() function, 201
  make_transpose_graph() function, 200
  transpose graph, 200
Kruskal, Joseph B., 161
Kruskal's algorithm, 161–164
  cost, 161
  kruskals() function, 162

L

links. *See* edges
local clustering coefficient, 24
lowest-cost path. *See* shortest path
lowest-cost path problem. *See* shortest-path problem

M

Manhattan distance, 354
Markov chain, 210. *See also* random walks
Markov model, 210
matching, 255–257, 263–273. *See also* bipartite matching problem
  maximal, 256
  maximum-cardinality, 256
  maximum-weight, 256
  perfect, 257
Matching class, 264–265
  add_edge() method, 265
  constructor, 264
  remove_edge() method, 265
matrix transpose, 200
maximum-flow problem, 229–254
  anti-parallel edges, 231, 253–254
  augmenting paths, 238–243
  augment_multisink_graph() function, 253
  augment_multisource_graph() function, 252
  for bipartite matching, 269–273
  capacity, 230–232
  flow, 230–231
  residual, 239–241
    backward, 235, 239, 241
    forward, 235, 239, 241

maximum-flow problem *(continued)*
    sink node, 230
        with multiple sinks, 252–253
    source node, 230
        with multiple sources, 251–252
maximum likelihood estimation, 221
mazes
    generation, 165–169
    representation, 165
        as a graph, 46
minimum spanning tree, 153–173
    Kruskal's algorithm, 161–164
    maze generation, 165–169
    number of edges, 154
    Prim's algorithm, 156–161
    single linkage clustering, 169–173
modifiable priority queue, 357–368. *See also* `PriorityQueue` class

N

negative cycles, 96
negative edge weight. *See also* edges
neighborhood subgraphs, 26–30
    `make_undirected_neighborhood_subgraph()` method, 27
neighbors, 19–22
    in directed graphs, 21–22
    `get_in_neighbors()` method, 21
    `get_neighbors()` method, 20
    `get_out_neighbors()` method, 21
    in-neighbor, 21
    out-neighbor, 21
    in undirected graphs, 20–21
`Node` class, 10–11
    `add_edge()` method, 11
    constructor, 10
    `get_edge_list()` method, 11
    `get_edge()` method, 10
    `get_neighbors()` method, 20
    `get_out_neighbors()` method, 21
    `get_sorted_edge_list()` method, 11
    `num_edges()` method, 10
    `remove_edge()` method, 11
nodes, 4. *See also* degree
    graphical representation, 4
    index, 9, 12
    neighbors, 19–22
NP-hard problem, 282, 299, 323

O

open-neighborhood subgraphs, 27
order index, 180
out-degree, 23
out-neighbors, 21

P

path planning, 70–73
paths, 31–42, 325–341
    `check_edge_path_valid()` function, 35
    `check_last_path_valid()` function, 39
    `check_node_path_valid()` function, 33
    `compute_path_cost_from_edges()` function, 40
    cost, 39–41, 90
        loop, 91
        negative edge weight, 91
        `PathCost()` expression, 90
    destination, 32
    as a list of edges, 34–36
    as a list of nodes, 33
    as a list of previous nodes, 36–39
    `make_node_path_from_last()` function, 37
    origin, 32
    and reachability, 41–42
    with repeated nodes, 32
`PGState` class, 81
`Point` class, 170, 354
post-order index, 199
preorder index, 180
Prim, R. C., 156
Prim's algorithm, 156, 161
    cost, 157
    `prims()` function, 158
`PriorityQueue` class, 361–368
    constructor, 362
    `dequeue()` method, 361, 366
    `_elements_inverted()` method, 363
    `enqueue()` method, 361, 365
    `get_priority()` method, 361–362
    `in_queue()` method, 361–362
    `is_empty()` method, 361–362
    `peek_top()` method, 362, 368

peek_top_priority() method, 362, 368
peek_top_value() method, 362, 368
_propagate_down() method, 365
_propagate_up() method, 364
size() method, 362
_swap_elements() method, 364
update_priority() method, 362, 367
uses
in A* search, 120
in Dijkstra's algorithm, 93
in greedy best-first search, 115
in Prim's algorithm, 158
puzzles, 75–88
river-crossing, 78–79, 81–88
slider, 79–80
solving with search, 85–88
states, 76, 81
state space graph, 78
constructing, 77, 80–85
Tower of Hanoi, 76–77

Q

queue, 65

R

randomized algorithm, 321–323
random walks, 207–226
absorbing nodes, 217
absorption probability, 217
absorption time, 217
choose_next_node() function, 215
choose_start() function, 224
estimate_graph_from_random_walks() function, 222
estimate_start_from_random_walks() function, 225
expected absorption time, 218
expected hitting time, 218
and games of chance, 214–215, 219–221
hitting probability, 217
hitting time, 217
is_valid_probability_graph() function, 211
learning, 221–226
random starting node, 224
random_walk() function, 216
self-loops, 208
simulating, 215–216, 224–225
stationary distribution, 218–219
time invariance, 210
transition matrix, 211
transition probability, 208, 210–211
reachability, 41–48
checking, 48, 195
in directed graphs, 193
get_reachable() function, 195
mutually reachable, 195
relaxation, 97
residual, 239–241
ResidualGraph class, 235–237, 240, 243
compute_total_flow() method, 237
constructor, 236
get_edge() method, 236
get_residual() method, 241
insert_edge() method, 236
min_residual_on_path() method, 243
update_along_path() method, 243
river-crossing puzzle, 78–79, 81–88
A* search, 124–126
create_prisoners_and_guards() function, 84
pg_generate_heuristic() function, 125
pg_neighbors() function, 83
pg_result_of_move() function, 82
PGState class, 81
pg_state_to_index_map() function, 85
solve_pg_bfs() function, 86
states, 78

S

self-loops, 22
in adjacency list representation, 22
in adjacency matrix representation, 22
and degree, 23
in random walks, 208
and undirected graphs, 22
Sharir, M., 199
shortest path, 90–91
in unweighted graphs, 68
in weighted graphs, 69

shortest-path problem, 89–110
  all-pairs, 102–103
  Bellman-Ford algorithm, 97–99
  Dijkstra's algorithm, 91–96
  Floyd-Warshall algorithm, 103–108
  negative edge weight, 96–97
single-linkage clustering, 169–173
  `Link` class, 171
  `Point` class, 170–171
  `single_linkage_clustering()` function, 171
sink node, 230
slider puzzle, 79–80
source node, 230
spanning tree, 154
spatial data, 353–355
  `build_graph_from_points()` function, 354
  Euclidean distance, 112, 354
  Manhattan distance, 354
  `Point` class, 354
state space graphs, 78
  constructing, 77, 80–85
stationary distribution, 218–219
strongly connected components, 193–205
  `check_strongly_connected()` function, 196
  `get_reachable()` function, 195
  Kosaraju-Sharir's algorithm, 199–204
strongly connected graphs, 194

T

Tarjan, Robert, 180, 204
task graphs, 138
time invariance, 210
topological sort, 135–151
  depth-first search, 143–147
    order of starting nodes, 146
    `topological_dfs()` function, 143
  detecting cycles, 147–148
    `check_cycle_kahns()` function, 148
  `is_topo_ordered()` function, 137
  Kahn's algorithm, 139
  `sort_forward_pointers()` function, 149
  use for ordering lists, 148–150
Tower of Hanoi puzzle, 76–77
transition matrix, 211
transition probability, 208, 210–211
  learning, 221–226
transpose graph, 200–201
  `make_transpose_graph()` function, 200
  of undirected graphs, 201
traveling salesperson problem, 331–334
  heuristics, 333
  `tsp_dfs()` function, 333
  `tsp_dfs_rec()` function, 332

U

undirected edges, 6, 10. *See also* edges
undirected graphs, 6, 12, 14
Union-Find, 161, 369–373
`UnionFind` class
  `are_disjoint()` method, 372
  constructor, 372
  `find_set()` method, 372
  `UnionFindNode` class, 371
  `union_sets()` method, 372
  uses
    in Kruskal's algorithm, 161, 163
    in maze generation, 167
    in single-linkage clustering, 171
unweighted graphs, 5, 9

V

$|V|$, 4
vertex cover, 315–321
  heuristics, 316
  `is_vertex_cover()` function, 315
  minimum, 316
  `minimum_vertex_cover_backtracking()` function, 319
  `minimum_vertex_cover_rec()` function, 318

using backtracking search, 318–321
using greedy search, 316–318
`vertex_cover_greedy_choice()` function, 316
`vertex_cover_greedy()` function, 317
vertexes. *See* nodes

W

walks, 32
weighted edges, 5. *See also* edges
weighted graphs, 5
`World` class, 127–130

Z

Zingaro, Daniel, 369