Gabriel Gonzalez’s Competitive Programming Bible

Copyright Gabriel Gonzalez 2018 ©

How to use this guide

The text in this document contains the heart of the algorithms. The helper classes and methods are not included because it would create too much clutter.

Keep in mind that you may have to edit the code depending on the context of the problem

Table of contents

Binary Search

Graph Coloring

Dijkstra

HeapSort

KnapSack

Selection Sort

Prims

Subset sum

Union Find

BFS on a grid

BFS on a graph

Topological sort

Elemental

0-1 BFS

Kruskals

Copyright Gabriel Gonzalez 2018 ©

Binary Search

**public static int** BS(**int**[] nums, **int** target) {

**int** max = nums.length; **int** min = -1;

**for**(**int** i=0; i < Math.*ceil*(Math.*log*(nums.length) / Math.*log*(2)); i++) {

**int** index = (max + min) / 2;

**if**(min + 1 == max)

**break**;

**if**(nums[index] == target)

**return** index;

**else if**(nums[index] > target)

max = index;

**else**

min = index; } **return** -1; }

Graph Coloring

**static int**[] *colors*;

**static int**[][] *adjMatrix*; **static int** *maxColors*;

**public static boolean** isSafe(**int** k, **int** c){

**for**(**int** i=0; i < *adjMatrix*.length; i++){

**if**(*adjMatrix*[k][i] == 1 && c == *colors*[i])

**return false**; } **return true**; }

**public static void** graphColor(**int** k){

**for**(**int** c=0; c < *maxColors*; c++){

**if**(*isSafe*(k,c)){

*colors*[k] = c; **if**(k == *colors*.length-1)

**return**; **else**

*graphColor*(k+1); } } }

Copyright Gabriel Gonzalez 2018 ©

Dijkstra

**class** edge {

**int** len; Dnode to; **public** edge( **int** l, Dnode n ){

len = l; to = n; } } **class** Dnode **implements** Comparable< Dnode > {

**boolean** visited = **false**; HashSet< edge > adj = **new** HashSet< edge >(); String id; **int** dist = Integer.***MAX\_VALUE***;

**public** Dnode( String s ){id = s;}

**public void** addEdge( Dnode a, **int** l ){

adj.add( **new** edge( l, a ) ); }

**public int** compareTo( Dnode n ) {

**return** dist - n.dist; }

} **public static int**[] shortestpath( Dnode[] nodes ) {

PriorityQueue< Dnode > q = **new** PriorityQueue< Dnode >(); Dnode start = nodes[0]; start.dist = 0; q.add(start);

**while**(!q.isEmpty()) {

Dnode curr = q.remove(); curr.visited = **true**;

**for**(edge e : curr.adj){

**if**(!e.to.visited){

e.to.dist = Math.*min*(e.to.dist, curr.dist + e.len); q.remove(e.to); q.add(e.to); } } }

**int**[] dist = **new int**[nodes.length];

**for**(**int** i=0; i < nodes.length; i++)

dist[i] = nodes[i].dist; **return** dist;

}

Copyright Gabriel Gonzalez 2018 ©

HeapSort

**public static int**[] heapSort(**int**[] nums){

**for**(**int** i=0; 2\*i + 2 < nums.length;i++){

**if**(nums[i] > nums[2\*i+1]){

**int** temp = nums[i]; nums[i] = nums[2\*i+1]; nums[2\*i+1] = temp; }**else if**(nums[i] > nums[2\*i+2]){ **int** temp = nums[i]; nums[i] = nums[2\*i+2]; nums[2\*i+2] = temp; } } **return** nums; }

KnapSack

**public static int** knapSack(**int** W(max weight), **int** wt[], **int** val[], **int** n(len)) {

**int** i, w; **int** K[][] = **new int**[n+1][W+1] **for** (i = 0; i <= n; i++) {

**for** (w = 0; w <= W; w++) {

**if** (i==0 || w==0)

K[i][w] = 0; **else if** (wt[i-1] <= w)

K[i][w] = Math.*max*(val[i-1] + K[i-1][w-wt[i-1]], K[i-1][w]); **else**

K[i][w] = K[i-1][w]; } } **return** K[n][W]; }

Selection Sort

**public static int**[] SelectionSort(**int**[] a){

**for**(**int** i=0; i < a.length-1; i++) {

**int** mindex=i; **for**(**int** j=i+1; j<a.length; j++){

**if**(a[j] < a[mindex])

mindex = j; } **if**(mindex != i) {

**int** t = a[i]; a[i] = a[mindex]; a[mindex]= t; } } **return** a; }

Copyright Gabriel Gonzalez 2018 ©

Prims

**public static int** prim(**int**[][] adjMat, **int** ind/\*start as 0\*/, **boolean**[] checked, **int** len, ArrayList<Integer> checkList){

checked[ind] = **true**; checkList.add(ind); **int** index = -1; **int** min = Integer.***MAX\_VALUE***; **for**(**int** row : checkList){

**for**(**int** c=0; c < checked.length; c++)

**if**(!checked[c] && adjMat[row][c] < min){

index = c; min = adjMat[row][c]; } } **if**(index == -1)

**return** len; **return** *prim*(adjMat,index, checked, len+min, checkList); }

Subset sum

**public static** ArrayList<Integer> findSubSetSum(**int** input[], **int** total) {

**boolean** T[][] = **new boolean**[input.length + 1][total + 1]; **for** (**int** i = 0; i <= input.length; i++) {

T[i][0] = **true**; } **for** (**int** i = 1; i <= input.length; i++)

**for** (**int** j = 1; j <= total; j++)

**if** (j - input[i - 1] >= 0)

T[i][j] = T[i - 1][j] || T[i - 1][j - input[i - 1]]; **else**

T[i][j] = T[i-1][j];

**if**(T[input.length][total]) {

**int** s = 0; ArrayList<Integer> set = **new** ArrayList<>(); **int** rr = input.length; **int** cc = total; **while**(s != total) {

**if**(rr-1 >= 0) {

**if**(T[rr-1][cc] == **false**) {

s += input[rr-1]; set.add(input[rr-1]); cc -= input[rr-1]; } rr--;

} } **return** set; } **return null**; }

Copyright Gabriel Gonzalez 2018 ©

Union Find

**public class** UnionFind {

**static int**[] *id*;

**public static int** getroot(**int** i){

**while**(i != *id*[i]){

*id*[i] = *id*[*id*[i]]; i = *id*[i]; } **return** i;

}

**public static void** main(String[] args) **throws** IOException {

**int**[] rank = **new int**[num]; *id* = **new int**[num];

**for**(**int** i=0; i < num; i++){

*id*[i] = i; rank[i] = 1; }

**while**( commands --> 0 ){

**if**(line[0].equals( "?" )){

**if**(*getroot*(a)==*getroot*(b))

out.println("yes"); **else**

out.println("no"); } **else**{

**int** i = *getroot*(a); **int** j = *getroot*(b); **if**(i == j)

**continue**; **if**(rank[i] < rank[j]){*id*[i] = j; rank[j] += rank[i];} **else**{*id*[j] = i; rank[i] += rank[j];} } } }

}

Copyright Gabriel Gonzalez 2018 ©

BFS on a grid

ArrayList<Point> queue = **new** ArrayList<>();

queue.add(start); dist[start.r][start.c] = 0;

**while** (!queue.isEmpty()) {

Point current = queue.remove(0);

**for** (**int** i = 0; i < 4; i++) {

**int** r = current.r + *R*[i]; **int** c = current.c + *C*[i];

**if** (r >= 0 && c >= 0 && r < rows && c < cols)

**if** (dist[r][c] == -1 && (map[r][c] == '#' || map[r][c] == 'E'))

{

dist[r][c] = 1 + dist[current.r][current.c]; queue.add(**new** Point(r , c)); } } }

BFS on a graph

ArrayList<Node> q = **new** ArrayList<Node>();

ArrayList<Node> visited = **new** ArrayList<Node>(); q.add(start); visited.add(start); **while**(!q.isEmpty()) {

Node curr = q.remove(0); visited.add(curr); **for**(Node n : curr.connections) {

n.distance = Math.*min*(n.distance, curr.distance+1); q.add(n); visited.add(n); }

}

Copyright Gabriel Gonzalez 2018 ©

Topological sort

**public class** TopologicaSort {

**public static** ArrayList< Node > DFS(Node n) {

Stack <Node> stack = **new** Stack< Node >(); ArrayList< Node > list = **new** ArrayList< Node >(); stack.push(n); loop : **while**(stack.size() > 0) {

Node k = stack.peek(); k.visited = **true**; **for**( Node l : k.connections) {

**if**(!l.visited) {

stack.push(l); **continue** loop; } } list.add(stack.pop()); } **return** list; }

**public static** ArrayList< Node > TopSort(ArrayList< Node > list) {

ArrayList< Node > sorted = **new** ArrayList< Node >();

**for**(Node n : list) {

**if**(!n.visited) {

ArrayList< Node > dfs = *DFS*(n); **for**(Node k : dfs) {

sorted.add(k); } } }

**return** sorted;

}

Copyright Gabriel Gonzalez 2018 ©

Elemental

**static boolean** *combinatioFound* = **false**;

**public static void** DFS(String curr, String goal, Node s){

curr += s.val; **if**(curr.equals(goal)){

*combinatioFound* = **true**;; **return**; } **for**(Node n : s.adj){

**if**(goal.startsWith(curr + n.val))

*DFS*(curr, goal, n); } }

**public static boolean** canCons(String s, ArrayList<Node> nodes){

**for**(Node n : nodes){

*DFS*("", s, n); **if**(*combinatioFound*){

*combinatioFound* = **false**; **return true**; } } **return false**; }

0-1 BFS

**while**(!q.isEmpty()) {

Point curr = q.remove(0); **for**(**int** i=0; i < 4; i++) {

**int** newR = curr.r + R[i]; **int** newC = curr.c + C[i]; **if**(*inBounds*(newR, newC, rows, cols) && dist[newR][newC] == -1) {

**if**(map[newR][newC] == '#') {

dist[newR][newC] = dist[curr.r][curr.c] + 1; q.add(**new** Point(newR, newC)); } **else** {

dist[newR][newC] = dist[curr.r][curr.c]; q.add(0, **new** Point(newR, newC)); } } } }

Copyright Gabriel Gonzalez 2018 ©

Kruskals

//Set up union find

Collections.*sort*(edges);

ArrayList<Edge> used = **new** ArrayList<Edge>();

K : **for**(Edge e : edges) {

**int** a = *getroot*(e.id1); **int** b = *getroot*(e.id2); **if**(a == b)

**continue** K; used.add(e); **if**(rank[a] < rank[b]){

*id*[a] = b; rank[b] += rank[a]; } **else**

*id*[b] = a; rank[a] += rank[b]; }

Copyright Gabriel Gonzalez 2018 ©