

Contest Templates - Index

1. 8 Queens - 1-36
2. DFS - 39-50
3. Total Divisors - 53-74
4. Fenwick Tree - 77 -112
5. Floyd Warshall - 115-126
6. isCycle (DFS Unidirected) - 129-151
7. Kadane's Algorithm - 154-163
8. Kruskal MST - 166- 237
9. MagicSquare (3x3) - 240-295
10. Maximum Sub Array Sum - 298 - 370
11. MST Tree From Graph - 373-443
12. nCR - 446-461
13. Segment Tree - 464-514
14. Sieve Prime - 517 -537
15. Structure with Compare Function - 540-553
16. TopSort - 556-569
17. Union Finder - 572-586

```

1 // 8Queens
2 pair < int, int > queen[9];
3 bool isvalid ( int row, int column ) {
4     for ( int i = 1; i < row; i++ ) {
5         if ( queen[i].ss == column || queen[i].ss - queen[i].ff == column - row ||
queen[i].ss + queen[i].ff == column + row )
6             return false;
7     }
8
9     return true;
10 }
11 int cases = 0;
12 void generate ( int row ) {
13     if ( row == 9 ) {
14         cout << endl << "Way N.O. = " << ++cases << endl << endl;
15
16         for ( int i = 1; i <= 8; i++ ) {
17             cout << "Row = " << i << " | Column = " << queen[i].ss << endl;
18         }
19
20         return;
21     }
22
23     FOR ( i, 1, 8 ) {
24         if ( isvalid ( row, i ) ) {
25             queen[row] = mp ( row, i );
26             generate ( row + 1 );
27         }
28     }
29 }
30
31 int main() {
32     freopen ( "in.txt", "r", stdin );
33     generate ( 1 );
34     return 0;
35 }
36 // 8 Queens ends
37
38
39 // DFS
40 void dfs ( int source, bool visited[], vi graph[] ) {
41     visited[source] = true;
42     int l = graph[source].size();
43     rep ( i, l ) {
44         int v = graph[source][i];
45
46         if ( !visited[v] )
47             dfs ( v, visited, graph );
48     }
49 }
50 // DFS ends
51
52
53 // Divisors
54 int numDiv ( int N ) {
55     int i = 0, factor = primes[i], ans = 1;
56
57     while ( factor * factor <= N ) {
58         int power = 0;
59
60         while ( N % factor == 0 ) {
61             N /= factor;
62             power++;
63         }
64
65         ans *= ( power + 1 );
66         factor = primes[++i];
67     }
68
69     if ( N != 1 )
70         ans *= 2;
71
72     return ans;
73 }
74 // Divisors End
75
76
77 //Fenwick Tree Begins
78 int t;
79 long tree[1234567];
80
81 int next ( int n ) {
82     return n + ( n & -n );
83 }

```

```

84 int prev ( int n ) {
85     return n - ( n & -n );
86 }
87
88 void update ( int i, long val ) {
89     while ( i <= t ) {
90         tree[i] += val;
91         i = next ( i );
92     }
93 }
94
95 long getsum ( int x ) {
96     long sum = 0;
97
98     while ( x > 0 ) {
99         sum += tree[x];
100         x = prev ( x );
101     }
102
103     return sum;
104 }
105
106 long sum ( int a, int b ) {
107     if ( a > b )
108         swap ( a, b );
109
110     return getsum ( b ) - getsum ( a - 1 );
111 }
112 //Fenwick Tree Ends
113
114
115 //Floyd Warshall Begins
116 //Initially mat 2D array should hold inf in all of its indexes
117 for ( int k = 0; k < n; k++ ) {
118     for ( int i = 0; i < n; i++ ) {
119         for ( int j = 0; j < n; j++ ) {
120             if ( mat[i][k] + mat[k][j] < mat[i][j] )
121                 mat[i][j] = mat[i][k] + mat[k][j];
122         }
123     }
124 }
125
126 //Floyd Warshall Ends
127
128
129 //isCycle (DFS Undirected) begins
130 vector <int> graph[123456];
131
132 // Initially int parent is infinity.
133 bool mark[123456];
134
135 bool iscycle ( int n, int parent ) {
136     mark[n] = true;
137     int length = graph[n].size();
138
139     for ( int i = 0; i < length; i++ ) {
140         int v = graph[n][i];
141
142         if ( !mark[v] ) {
143             if ( iscycle ( v, n ) )
144                 return true;
145             } else if ( v != parent )
146                 return true;
147         }
148
149     return false;
150 }
151 //isCycle ends
152
153
154 // Kadane Algorithm's Begins
155 int best = 0, sum = 0;
156
157 for ( int k = 0; k < n; k++ ) {
158     sum = max ( array[k], sum + array[k] );
159     best = max ( best, sum );
160 }
161
162 cout << best << "\n";
163 // Kadane Algorithm's ends
164
165
166 //Kruskal MST Begins
167 int parent[1234567];

```

```

168
169 void setInitialParent() {
170     for ( int i = 0; i < 1234567; i++ )
171         parent[i] = i;
172 }
173
174 int fin ( int r ) {
175     if ( parent[r] == r )
176         return r;
177
178     return parent[r] = fin ( parent[r] );
179 }
180
181 void unio ( int x, int y ) {
182     int u = fin ( x );
183     int v = fin ( y );
184     parent[u] = parent[v];
185 }
186
187 struct node {
188     int u, v, cost;
189
190     node ( int x, int y, int z ) {
191         u = x, v = y, cost = z;
192     };
193
194     bool operator < ( node other ) const {
195         return cost < other.cost;
196     };
197 };
198
199
200 int main() {
201     //freopen ("in.txt", "r", stdin);
202     //freopen ("out.txt", "w", stdout);
203     //ios_base::sync_with_stdio(0); cin.tie(0);
204     int m, n; //m is number of nodes & n is number of edges
205
206     while ( scii ( m, n ) && m != 0 && n != 0 ) {
207         setInitialParent();
208         int x, y, z;
209         vector < node > edge;
210
211         for ( int i = 0; i < n; i++ ) {
212             scii ( x, y );
213             sci ( z );
214             edge.push_back ( node ( x, y, z ) );
215         }
216
217         sort ( edge.begin(), edge.end() );
218         int realCost = 0;
219
220         for ( int i = 0; i < n; i++ ) {
221             int u = edge[i].u;
222             int v = edge[i].v;
223             int cost = edge[i].cost;
224             x = fin ( u ), y = fin ( v );
225
226             if ( x != y ) {
227                 unio ( x, y );
228                 realCost += cost;
229             }
230         }
231
232         printf ( "%d\n", realCost );
233     }
234
235     return 0;
236 }
237 // Kruskal MST ENDS
238
239
240 // MagicSquare(3x3) the sum of any row, column, or diagonal = 15
241 #include <bits/stdc++.h>
242
243 using namespace std;
244
245 int magic [9][3][3];
246
247 void process ( int i, int j, int num, int index ) {
248     if ( !magic[index][i][j] ) {
249         magic[index][i][j] = num;
250
251         if ( num % 3 == 0 ) {

```

```

252         j++;
253         j = j % 3;
254     } else {
255         i++, j--;
256         i = i % 3;
257         j = ( j + 3 ) % 3;
258     }
259
260     num += 1;
261     process ( i, j, num, index );
262 }
263 }
264
265 void generateMagicSquare() {
266     memset ( magic, 0, sizeof magic );
267     int index = 0;
268
269     for ( int i = 0; i < 3; i++ ) {
270         for ( int j = 0; j < 3; j++ ) {
271             process ( i, j, 1, index );
272             index++;
273         }
274     }
275
276     for ( int i = 0; i < 9; i++ ) {
277         cout << "Index = " << i << endl;
278
279         for ( int j = 0; j < 3; j++ ) {
280             for ( int k = 0; k < 3; k++ ) {
281                 cout << magic[i][j][k] << " ";
282             }
283
284             cout << endl;
285         }
286
287         cout << endl << endl;
288     }
289 }
290
291 int main() {
292     generateMagicSquare();
293     return 0;
294 }
295 // MagicSquare ends
296
297
298 // Max Sub Array Sum Begins
299 #include<cstdio>
300 #include<cmath>
301 #include<iostream>
302 #include<climits>
303
304 using namespace std;
305
306 int max ( int a, int b ) {
307     return ( ( a > b ) ? a : b );
308 }
309
310 int max ( int a, int b, int c ) {
311     return max ( max ( a, b ), c );
312 }
313
314 int MaxCrossingArray ( int a[], int b, int m, int c ) {
315     int left_sum = INT_MIN;
316     int sum = 0;
317
318     for ( int i = m; i >= b; i-- ) {
319         sum += a[i];
320
321         if ( sum > left_sum ) {
322             left_sum = sum;
323         }
324     }
325
326     int right_sum = INT_MIN;
327     sum = 0;
328
329     for ( int i = m + 1; i <= c; i++ ) {
330         sum += a[i];
331
332         if ( sum > right_sum )
333             right_sum = sum;
334     }
335

```

```

336     return ( left_sum + right_sum );
337 }
338
339 int MaxSubArray ( int a[], int c, int d ) {
340     if ( c == d )
341         return a[c];
342
343     int m = ( c + d ) / 2;
344     return max ( MaxSubArray ( a, c, m ), MaxSubArray ( a, m + 1, d ), MaxCrossingArray
( a, c, m, d ) );
345 }
346
347 int main() {
348     while ( true ) {
349         int a;
350         scanf ( "%d", &a );
351
352         if ( a == 0 )
353             break;
354
355         int b[100005];
356
357         for ( int j = 0; j < a; j++ )
358             cin >> b[j];
359
360         int max = MaxSubArray ( b, 0, a - 1 );
361
362         if ( max > 0 )
363             printf ( "The maximum winning streak is %d.\n", max );
364         else
365             printf ( "Losing streak.\n" );
366     }
367
368     return 0;
369 }
370 // Max Sub Array Ends
371
372 // MST tree from Graph
373 int parent[1234567];
374
375 void setInitialParent() {
376     for ( int i = 0; i < 1234567; i++ )
377         parent[i] = i;
378 }
379
380 int fin ( int r ) {
381     if ( parent[r] == r )
382         return r;
383
384     return parent[r] = fin ( parent[r] );
385 }
386
387 void unio ( int x, int y ) {
388     int u = fin ( x );
389     int v = fin ( y );
390     parent[u] = parent[v];
391 }
392
393 struct node {
394     int u, v, cost;
395
396     node ( int x, int y, int z ) {
397         u = x, v = y, cost = z;
398     };
399
400     bool operator < ( node other ) const {
401         return cost < other.cost;
402     };
403 };
404
405 int main() {
406     //freopen ( "in.txt", "r", stdin );
407     //freopen ( "out.txt", "w", stdout );
408     //ios_base::sync_with_stdio(0); cin.tie(0);
409     int m, n; //m is number of nodes & n is number of edges
410
411     while ( scii ( m, n ) && m != 0 && n != 0 ) {
412         setInitialParent();
413         int x, y, z;
414         vector < node > edge;
415         vector < pair < int, int > > graph [110];
416     }
417 }

```

```

419     for ( int i = 0; i < n; i++ ) {
420         scii ( x, y );
421         sci ( z );
422         edge.push_back ( node ( x, y, z ) );
423     }
424
425     sort ( edge.begin(), edge.end() );
426
427     for ( int i = 0; i < n; i++ ) {
428         int u = edge[i].u;
429         int v = edge[i].v;
430         int cost = edge[i].cost;
431         x = fin ( u ), y = fin ( v );
432
433         if ( x != y ) {
434             unio ( x, y );
435             graph[u].push_back ( make_pair ( v, cost ) );
436             graph[v].push_back ( make_pair ( u, cost ) );
437         }
438     }
439 }
440
441 return 0;
442 }
443 // MST tree from Graph ENDS
444
445 // NCR Begins
446 L dp[110][110];
447
448 L nCr ( int n, int r ) {
449     if ( n == r || r == 0 )
450         return 1;
451
452     if ( r == 1 )
453         return n;
454
455     if ( dp[n][r] )
456         return dp[n][r];
457
458     return dp[n][r] = nCr ( n - 1, r - 1 ) + nCr ( n - 1, r );
459 }
460 // NCR ends
461
462 //Segment Tree Begins
463 int A[1234567];
464 int Tree[12345678];
465
466 void buildTree ( int p, int left, int right ) {
467     if ( left == right ) {
468         Tree[p] = A[left];
469     } else {
470         int mid = ( left + right ) / 2;
471         buildTree ( p << 1, left, mid );
472         buildTree ( ( p << 1 ) + 1, mid + 1, right );
473         Tree[p] = Tree[p << 1] + Tree[ ( p << 1 ) + 1];
474     }
475 }
476
477 int fquery ( int p, int left, int right, int x, int y ) {
478     if ( x > right || y < left )
479         return -1;
480
481     if ( left >= x && right <= y )
482         return Tree[p];
483
484     int mid = ( left + right ) / 2;
485     int p1 = fquery ( p << 1, left, mid, x, y );
486     int p2 = fquery ( ( p << 1 ) + 1, mid + 1, right, x, y );
487
488     if ( p1 == -1 )
489         return p2;
490
491     if ( p2 == -1 )
492         return p1;
493
494     return p1 + p2;
495 }
496
497 void updateTree ( int p, int left, int right, int x, int y ) {
498     if ( left == right ) {
499         A[left] = y;
500         Tree[p] = y;
501     }
502 }

```

```

503     } else {
504         int mid = ( left + right ) / 2;
505
506         if ( x >= left && x <= mid )
507             updateTree ( p << 1, left, mid, x, y );
508         else
509             updateTree ( ( p << 1 ) + 1, mid + 1, right, x, y );
510
511         Tree[p] = Tree[p << 1] + Tree[ ( p << 1 ) + 1];
512     }
513 }
514 //Segment Tree ends
515
516 // Seive Prime Begins
517 int prime[Local];
518 bool chk[Global];
519 int koto;
520 void pr() {
521     prime[koto++] = 2;
522     int n = sqrt ( Global ) + 1;
523
524     for ( int i = 3; i <= n; i += 2 ) {
525         if ( !chk[i] ) {
526             for ( int j = i * i; j < Global; j += i * 2 ) {
527                 chk[j] = true;
528             }
529         }
530     }
531
532     for ( int i = 3; i < Global; i += 2 )
533         if ( !chk[i] )
534             prime[koto++] = i;
535 }
536 // Seive prime Ends
537
538 // Structure with Compare Function
539 struct node {
540     int u, v, cost;
541
542     node ( int x, int y, int z ) {
543         u = x, v = y, cost = z;
544     };
545
546     bool operator < ( node other ) const {
547         return cost < other.cost;
548     };
549 };
550 // Structure with Compare Function ENDS
551
552 // Top Sort Begins
553 vector < int > lst;
554
555 void topsort ( int source, bool visited[], vector < int > graph[] ) {
556     visited[source] = true;
557     rep ( i, ( int ) graph[source].size() )
558         if ( !visited[graph[source][i]] )
559             topsort ( graph[source][i], visited, graph );
560     lst.pb ( source );
561 }
562 // Top Sort Ends
563
564 //Union Finder Begins
565 int parent[200010];
566
567 int fin ( int r ) {
568     if ( parent[r] == r )
569         return r;
570
571     return parent[r] = fin ( parent[r] );
572 }
573
574 void unio ( int x, int y ) {
575     int u = fin ( x );
576     int v = fin ( y );
577     parent[u] = parent[v];
578 }

```