1 # Travelling Salesman --------->

2

3

4 int adj[20][20];

5 int vis[20];

6 int solve(int cnt , int last)

7 {

8 if (cnt == n)

9 {

10 return adj[last][0];

11 }

12 int ret = inf;

13 for (int i = 1; i < n; i++)

14 {

15 if (vis[i] == 0)

16 {

17 vis[i] = 1;

18 ret = min(ret , solve(cnt + 1 , i) + adj[last][i] );

19 vis[i] = 0;

20 }

21 }

22 return ret;

23 }

24

25 int main()

26 {

27 int t;

28 cin >> t;

29 while (t--)

30 {

31 cin >> n;

32 for (int i = 0; i < n; i++) 33 {

34 vis[i] = 0;

35 for (int j = 0; j < n; j++) 36 {

37 cin >> adj[i][j];

38 }

39 }

40

41 ans = solve(1, 0);

42 cout << ans << endl;

43 }

44

45 }

46

47

48 # Flip Column ----------------->

49

50

51 int arr[16][16];

52

53 void flip(int i) {

54 for (int j = 0; j < n; j++)

55 if (arr[j][i])

56 arr[j][i] = 0;

57 else

58 arr[j][i] = 1;

59 }

60

61

62 void solve(int cnt)

63 {

64 if (cnt == k)

65 {

66 int counT = 0 , row = 0; 67 for (int i = 0; i < n; i++) 68 {

69 counT = 0 ;

70 for (int j = 0; j < m; j++) 71 {

72 if ( arr[i][j] == 1 ) 73 counT++;

74 }

75 if (counT == m)

76 row++;

77 }

78

79 ans = max(ans , row); 80 return ;

81 }

82

83 for (int jj = 0; jj < m; jj++) 84 {

85 flip(jj);

86 solve(cnt + 1);

87 flip(jj);

88 }

89 return;

90 }

91

92 int main()

93 {

94 int t = 1;

95 //cin >> t;

96 while (t--)

97 {

98 cin >> n >> m >> k;

99 for (int i = 0; i < n; i++) 100 {

101 for (int j = 0; j < m; j++) 102 {

103 cin >> arr[i][j]; 104 }

105 }

106

107 ans = 0;

108 solve(0);

109 cout << ans << endl;

110 }

111

112 }

113

114

115 # Endoscope ------------------->

116

117 int a[50][50],n,m;

118 int vis[50][50]={0};

119

120 void DFS(int xpos,int ypos,int rem\_len){

121 if(xpos<0 || xpos>=n || ypos<0 || ypos>=m || rem\_len==0) 122 return;

123 vis[xpos][ypos]=1;

124 if(a[xpos][ypos] == 1){

125

126 if((xpos!=0) && (a[xpos-1][ypos] ==2 || a[xpos-1][ypos] ==5 || a[xpos-1][ypos] ==6 ||

a[xpos-1][ypos] ==1)) //up

127 DFS(xpos-1, ypos, rem\_len-1);

128

129 if((xpos!=n-1) && (a[xpos+1][ypos] ==2 || a[xpos+1][ypos] ==4 || a[xpos+1][ypos] ==7 ||

a[xpos+1][ypos] ==1)) //down

130 DFS(xpos+1, ypos, rem\_len-1);

131

132 if((ypos!=0)&& (a[xpos][ypos-1] ==3 || a[xpos][ypos-1] ==4 || a[xpos][ypos-1] ==5 ||

a[xpos][ypos-1] ==1)) //left

133 DFS(xpos, ypos-1, rem\_len-1);

134

135 if((ypos!=m-1) && (a[xpos][ypos+1] ==3 || a[xpos][ypos+1] ==6 || a[xpos][ypos+1] ==7 ||

a[xpos][ypos+1] ==1)) //right

136 DFS(xpos, ypos+1, rem\_len-1);

137

138 }

139 else if(a[xpos][ypos] == 2)

140 {

141 if((xpos!=0) && (a[xpos-1][ypos] ==1 || a[xpos-1][ypos] ==5 || a[xpos-1][ypos] ==6 ||

a[xpos-1][ypos] ==2)) //up

142 DFS(xpos-1, ypos, rem\_len-1);

143

144 if((xpos!=n-1) && (a[xpos+1][ypos] ==1 || a[xpos+1][ypos] ==4 || a[xpos+1][ypos] ==7 ||

a[xpos+1][ypos] ==2)) //down

145 DFS(xpos+1, ypos, rem\_len-1);

146 }

147 else if(a[xpos][ypos] == 3)

148 {

149 if((ypos!=0)&& (a[xpos][ypos-1] ==1 || a[xpos][ypos-1] ==4 || a[xpos][ypos-1] ==5 ||

a[xpos][ypos-1] ==3)) //left

150 DFS(xpos, ypos-1, rem\_len-1);

151

152 if((ypos!=m-1) && (a[xpos][ypos+1] ==1 || a[xpos][ypos+1] ==6 || a[xpos][ypos+1] ==7 ||

a[xpos][ypos+1] ==3)) //right

153 DFS(xpos, ypos+1, rem\_len-1); 154 }

155 else if(a[xpos][ypos] == 4)

156 {

157 if((xpos!=0) && (a[xpos-1][ypos] ==1 || a[xpos-1][ypos] ==2 || a[xpos-1][ypos] ==5 || a[xpos-1][ypos] ==6)) //up

158 DFS(xpos-1, ypos, rem\_len-1); 159

160 if((ypos!=m-1) && (a[xpos][ypos+1] ==1 || a[xpos][ypos+1] ==3 || a[xpos][ypos+1] ==6 || a[xpos][ypos+1] ==7)) //right

161 DFS(xpos, ypos+1, rem\_len-1); 162 }

163 else if(a[xpos][ypos] == 5)

164 {

165 if((xpos!=n-1) && (a[xpos+1][ypos] =1 || a[xpos+1][ypos] ==2 || a[xpos+1][ypos] ==7 || a[xpos+1][ypos] ==4)) //down

166 DFS(xpos+1, ypos, rem\_len-1); 167

168 if((ypos!=m-1) && (a[xpos][ypos+1] ==1 || a[xpos][ypos+1] ==3 || a[xpos][ypos+1] ==6 || a[xpos][ypos+1] ==7)) //right

169 DFS(xpos, ypos+1, rem\_len-1); 170 }

171 else if(a[xpos][ypos] == 6)

172 {

173 if((xpos!=n-1) && (a[xpos+1][ypos] ==1 || a[xpos+1][ypos] ==2 || a[xpos+1][ypos] ==7 || a[xpos+1][ypos] ==4)) //down

174 DFS(xpos+1, ypos, rem\_len-1); 175

176 if((ypos!=0)&& (a[xpos][ypos-1] ==1 || a[xpos][ypos-1] ==3 || a[xpos][ypos-1] ==5 || a[xpos][ypos-1] ==4)) //left

177 DFS(xpos, ypos-1, rem\_len-1); 178 }

179 else if(a[xpos][ypos] == 7)

180 {

181 if((xpos!=0) && (a[xpos-1][ypos] ==1 || a[xpos-1][ypos] ==2 || a[xpos-1][ypos] ==5 || a[xpos-1][ypos] ==6)) //up

182 DFS(xpos-1, ypos, rem\_len-1); 183

184 if((ypos!=0)&& (a[xpos][ypos-1] ==1 || a[xpos][ypos-1] ==3 || a[xpos][ypos-1] ==4 || a[xpos][ypos-1] ==5)) //left

185 DFS(xpos, ypos-1, rem\_len-1); 186 }

187 }

188

189

190 int main() {

191 int t,i,j,k,x,y,l;

192 cin>>t;

193 while(t--){

194

195 cin>>n>>m>>x>>y>>l;

196 for(i=0;i<n;i++){

197 for(j=0;j<m;j++)

198 cin>>a[i][j];

199 }

200

201 DFS(x,y,l);

202

203 int count=0;

204 for(i=0;i<n;i++){

205 for(j=0;j<m;j++){

206 if(vis[i][j]==1){

207 count++;

208 vis[i][j]=0;

209 }

210 }

211 }

212 cout<<count<<endl;

213 }

214 }

215

216

217

218

219 # Kim Refrigerator

220

221

222 int main()

223 {

224 fastio;

225 t = 10;

226 //cin >> t;

227 while (t--)

228 {

229 counT = inf;

230 cin >> n ;

231 vector<pair<int, int>>vc;

232 vector<int>per;

233

234 int homea , homeb , offa , offb ; 235 cin >> offa >> offb >> homea >> homeb; 236 for (int i = 0; i < n; i++)

237 {

238 cin >> a >> b;

239 per.push\_back(i + 1);

240 vc.push\_back({a, b});

241 }

242

243 do

244 {

245 sum = 0 ;

246 pair<int, int> pp = {offa , offb}; 247 for (int i = 0; i < per.size(); i++) {

248 sum += abs(vc[per[i] - 1].first - pp.first) + abs(vc[per[i] - 1].second - pp.second); 249 pp = vc[per[i] - 1];

250 }

251 sum += abs(homea - pp.first) + abs(homeb - pp.second);

252

253 // dbg(per);

254

255 counT = min(counT , sum);

256 } while ( (next\_permutation(per.begin(), per.end())) );

257

258 cout << "# " << ++cs << ' ' << counT << endl; 259

260

261 }

262 }

263

264

265

266

267

268 # Warmholes -------------------->

269

270

271 int mask[10], w[10][5], f = 0;

272 int distance(int sx, int sy, int dx, int dy) { 273 int xd = abs(dx - sx);

274 int yd = abs(dy - sy);

275 return (xd + yd);

276 }

277

278 void cal(int sx, int sy, int dx, int dy, int dis) 279 {

280 ans = min(ans, distance(sx, sy, dx, dy) + dis); 281 for (int i = 0; i < n; i++)

282 {

283 if (mask[i] == 0)

284 {

285 mask[i] = 1;

286

287 int temp = distance(sx, sy, w[i][0], w[i][1]) + dis + w[i][4];

288 cal(w[i][2], w[i][3] , dx , dy , temp); 289

290 temp = distance(sx, sy, w[i][2], w[i][3]) + dis + w[i][4];

291 cal(w[i][0], w[i][1] , dx , dy , temp); 292

293 mask[i] = 0 ;

294 }

295 }

296

297 }

298

299 int main()

300 {

301 fastio;

302 t = 10;

303 cin >> t;

304 while (t--)

305 {

306

307 cin >> n;

308 int sx, sy, dx, dy;

309 cin >> sx >> sy >> dx >> dy;

310

311 for (int i = 0; i < n; i++) {

312 mask[i] = 0;

313 for (int j = 0; j < 5; j++) {

314 cin >> w[i][j];

315 }

316 }

317 ans = 999999;

318 cal(sx, sy, dx, dy, 0);

319 cout << ans << endl;

320

321

322 }

323 }

324

325

326

327

328 # Burst Balloons Optimally -----------------> 329

330 ll arr[15] , vis[15] ;

331

332 void func(ll cnt , ll sum)

333 {

334 if (cnt == n)

335 {

336 Max = max(Max , sum);

337 return ;

338 }

339

340 ll new\_ans = sum;

341 for (int i = 0; i < n; i++)

342 {

343 if (vis[i] == 0)

344 {

345 vis[i] = 1;

346 ll lf = 0 , rf = 0 , lf\_val = 1 , rf\_val = 1; 347

348 for (j = i - 1; j >= 0; j--)

349 {

350 if (vis[j] == 0)

351 {

352 lf = 1;

353 lf\_val = arr[j];

354 break;

355 }

356 }

357 for (j = i + 1; j < n; j++) 358 {

359 if (vis[j] == 0)

360 {

361 rf = 1;

362 rf\_val = arr[j]; 363 break;

364 }

365 }

366

367

368 if (lf == 0 and rf == 0) 369 sum += arr[i];

370 else sum = sum + (lf\_val \* rf\_val); 371

372 func(cnt + 1 , sum);

373 vis[i] = 0;

374 sum = new\_ans;

375

376 }

377 }

378 return ;

379 }

380

381 int main()

382 {

383 fastio;

384

385 cin >> n;

386 for (int i = 0; i < n; i++)

387 {

388 cin >> arr[i];

389 vis[i] = 0;

390 }

391

392 Max = -inf;

393 func(0, 0);

394

395 cout << Max << ln;

396 }

397

398

399

400

401 # Fisherman

402

403 int gates[3];

404 int fisherman[3];

405 int visited[20];

406

407 void permut(int visited[] , int l , int r) 408 {

409 if (l == r)

410 {

411

412 int i, j, k, dist = 0;

413 for (i = 0; i < fisherman[0]; i++) { 414 dist = dist + abs(visited[i] - gates[0]) + 1 ; 415 }

416 for (j = 0; j < fisherman[1]; j++) { 417 dist = dist + abs(visited[i] - gates[1]) + 1 ; 418 i += 1;

419 }

420 for (k = 0; k < fisherman[2]; k++) { 421 dist = dist + abs(visited[i] - gates[2]) + 1 ; 422 i += 1;

423 }

424

425 counT = min(counT , dist);

426

427 return ;

428

429

430 }

431 else

432 {

433 for (int i = l; i <= r; i++) {

434 swap(visited[i], visited[l]);

435 permut(visited, l + 1, r);

436 swap(visited[i], visited[l]);

437 }

438 }

439 }

440

441 int main() {

442 cin >> n;

443 for (int i = 0; i < 3; i++)

444 cin >> gates[i];

445 for (int i = 0; i < 3; i++)

446 cin >> fisherman[i];

447

448 for (int i = 0; i < n; i++)

449 visited[i] = i + 1;

450

451 permut(visited, 0, n - 1);

452 cout << counT << endl;

453

454 }

455

456 # Aeroplane Bombing

457

458 ll arr[12][12];

459

460 void solve(int row , int col , int temp , int &ans , int bomb , int effect)

461 {

462 if (row < 0)

463 {

464 ans = max(ans , temp);

465 return;

466 }

467

468 for (int i = -1 ; i <= 1 ; i++)

469 {

470 if ( (col + i) < 0 or (col + i) > 4 ) continue; 471

472 if (arr[row][col + i] == 1 or arr[row][col + i] == 0) /// no enemy

473 {

474 if (bomb == 0)

475 solve(row - 1, col + i, temp + arr[row][col + i] , ans , bomb , effect - 1 );

476 else

477 solve(row - 1, col + i, temp + arr[row][col + i] , ans , bomb , effect );

478 }

479 else

480 {

481 if (bomb == 0)

482 {

483 if (effect > 0)

484 {

485 solve(row - 1, col + i, temp, ans , bomb , effect - 1 );

486 }

487 }

488 else

489 solve(row - 1, col + i, temp , ans , 0 , 5 );

490 }

491 }

492 return ;

493

494 }

495

496 int main()

497 {

498 fastio;

499 t = 1;

500 cin >> t;

501 while (t--)

502 {

503 cin >> n; m = 5;

504

505 f0(i, n)

506 {

507 f0(j, m)

508 {

509 cin >> arr[i][j];

510 }

511 }

512

513 int ans = 0 ;

514 solve(n - 1, 2, 0, ans , 1 , 0);

515 cout << "#" << ++cs << ' ' << ans << endl; 516

517 }

518 }

519

520

521

522 # Rock Climbing

523

524

525 ll arr[12][12];

526

527 void solve(int row , int col , int temp , int &ans , int bomb , int effect)

528 {

529 if (row < 0)

530 {

531 ans = max(ans , temp);

532 return;

533 }

534

535 for (int i = -1 ; i <= 1 ; i++)

536 {

537 if ( (col + i) < 0 or (col + i) > 4 ) continue; 538

539 if (arr[row][col + i] == 1 or arr[row][col + i] == 0) /// no enemy

540 {

541 if (bomb == 0)

542 solve(row - 1, col + i, temp + arr[row][col + i] , ans , bomb , effect - 1 );

543 else

544 solve(row - 1, col + i, temp + arr[row][col + i] , ans , bomb , effect );

545 }

546 else

547 {

548 if (bomb == 0)

549 {

550 if (effect > 0)

551 {

552 solve(row - 1, col + i, temp, ans , bomb , effect - 1 );

553 }

554 }

555 else

556 solve(row - 1, col + i, temp , ans , 0 , 5 );

557 }

558 }

559 return ;

560

561 }

562

563 int main()

564 {

565 fastio;

566 t = 1;

567 cin >> t;

568 while (t--)

569 {

570 cin >> n; m = 5;

571

572 f0(i, n)

573 {

574 f0(j, m)

575 {

576 cin >> arr[i][j];

577 }

578 }

579

580 int ans = 0 ;

581 solve(n - 1, 2, 0, ans , 1 , 0); 582 cout << "#" << ++cs << ' ' << ans << endl; 583

584 }

585 }

586

587

588

589

590

591 # Sum of Nodes at Kth Level

592

593

594 int main()

595 {

596 string s;

597 cin >> n >> s;

598 sum = 0 , counT = 0 ;

599 stack<pair<char , ll>>st;

600 for (int i = 0; i < s.size(); i++) 601 {

602 if (s[i] != ')')

603 {

604 if (s[i] == '(')

605 {

606 st.push({s[i], counT}); 607 counT++;

608 }

609 else

610 {

611 st.push({s[i], inf}); 612 }

613 }

614 else

615 {

616 string num = "";

617 // dbg( i , st.size());

618 while (st.top().F != '(') 619 {

620 if ( (st.top().F - '0') >= 0 and (st.top().F - '0') <= 9 )

621 {

622 num += st.top().F; 623 }

624 st.pop();

625 }

626 reverse(all(num));

627

628 int jog = 0;

629 f0(j, num.size())

630 jog = jog \* 10 + (num[j] - '0'); 631

632 // dbg(num , jog); 633

634 counT = st.top().S;

635 st.pop();

636 if (counT == n)

637 sum += jog;

638

639 }

640

641 }

642

643 cout << sum << endl;

644

645

646

647 }

648

649

650

651

652

653 # Detect CYcle and print minimum sum 654

655 vector<pll>vc;

656 vector<ll>graph[100];

657

658 vector<int>parent , color ;

659 int start\_cycle , end\_cycle ;

660

661 bool dfs(int node)

662 {

663 color[node] = 1;

664 for (auto it : graph[node])

665 {

666 if (color[it] == 0)

667 {

668 parent[it] = node;

669 if ( dfs(it) ) return true; 670 }

671 else if (color[it] == 1) 672 {

673 start\_cycle = it;

674 end\_cycle = node;

675 return true;

676 }

677

678 }

679 color[node] = 2;

680 return false;

681 }

682

683 int main()

684 {

685 fastio;

686 cin >> n >> m;

687 f0(i, 2 \* m)

688 {

689 cin >> a >> b ;

690 vc.pb({a, b});

691 }

692 Max = (1 << n) - 1;

693 counT = inf;

694 vector<ll>res;

695

696 i = Max;

697 for (i = 0; i <= Max; i++) 698 {

699

700 map<ll, ll>mp;

701 vector<ll>new\_node;

702

703 for (j = 0; j < n; j++) 704 {

705 if (checkBit(i, j)) 706 {

707 int node = j + 1; 708 mp[node]++;

709 }

710 }

711

712 f1(j, n)

713 {

714 graph[j].clear(); 715 }

716 for (auto it : vc)

717 {

718 if (mp[it.F] and mp[it.S]) 719 {

720 graph[it.F].pb(it.S); 721 new\_node.pb(it.F); 722 new\_node.pb(it.S); 723 }

724 }

725

726

727

728 /// Find Cycle

729 color.assign(n + 1 , 0); 730 parent.assign(n + 1 , -1); 731 start\_cycle = -1;

732

733 for (auto it : new\_node)

734 {

735 if (color[it] == 0 and dfs(it)) 736 break;

737 }

738

739

740 if (start\_cycle != -1)

741 {

742

743 vector<int>cycle;

744 ll cost = 0;

745 for (int v = end\_cycle ; v != start\_cycle ; v = parent[v])

746 {

747 cycle.pb(v);

748 cost += v;

749 }

750 cycle.push\_back(start\_cycle);

751

752

753 if (counT > cost)

754 {

755 sort(all(cycle));

756 for (auto it : cycle)

757 res.pb(it);

758 counT = cost;

759 }

760

761 }

762 }

763

764

765 for (auto it : res)

766 cout << it << ' ';

767 cout << endl;

768 }

769

770

771

772

773

774 # cycle undirected graph

775

776 int n;

777 vector<vector<int>> adj;

778 vector<bool> visited;

779 vector<int> parent;

780 int cycle\_start, cycle\_end;

781

782 bool dfs(int v, int par) { // passing vertex and its parent vertex

783 visited[v] = true;

784 for (int u : adj[v]) {

785 if(u == par) continue; // skipping edge to parent vertex

786 if (visited[u]) {

787 cycle\_end = v;

788 cycle\_start = u;

789 return true;

790 }

791 parent[u] = v;

792 if (dfs(u, parent[u]))

793 return true;

794 }

795 return false;

796 }

797

798 void find\_cycle() {

799 visited.assign(n, false);

800 parent.assign(n, -1);

801 cycle\_start = -1;

802

803 for (int v = 0; v < n; v++) {

804 if (!visited[v] && dfs(v, parent[v])) 805 break;

806 }

807

808 if (cycle\_start == -1) {

809 cout << "Acyclic" << endl;

810 } else {

811 vector<int> cycle;

812 cycle.push\_back(cycle\_start);

813 for (int v = cycle\_end; v != cycle\_start; v = parent[v])

814 cycle.push\_back(v);

815 cycle.push\_back(cycle\_start);

816 reverse(cycle.begin(), cycle.end()); 817

818 cout << "Cycle found: ";

819 for (int v : cycle)

820 cout << v << " ";

821 cout << endl;

822 }

823 }

824

825

826

827

828

829

830 # bipartite

831

832 bool isBipartite(int G[][V], int src)

833 {

834

835 int colorArr[V];

836 for (int i = 0; i < V; ++i)

837 colorArr[i] = -1;

838

839 // Assign first color to source

840 colorArr[src] = 1;

841

842 queue <int> q;

843 q.push(src);

844

845 // Run while there are vertices

846 // in queue (Similar to BFS)

847 while (!q.empty())

848 {

849 // Dequeue a vertex from queue ( Refer http://goo.gl/35oz8 )

850 int u = q.front();

851 q.pop();

852

853 // Return false if there is a self-loop 854 if (G[u][u] == 1)

855 return false;

856

857 for (int v = 0; v < V; ++v)

858 {

859

860 if (G[u][v] && colorArr[v] == -1) 861 {

862 colorArr[v] = 1 - colorArr[u]; 863 q.push(v);

864 }

865

866 else if (G[u][v] && colorArr[v] == colorArr[u]) 867 return false;

868 }

869 }

870

871 return true;

872 }

873

874

875

876

877 # Men's Restroom

878

879 #include<iostream>

880 #include<bits/stdc++.h>

881 using namespace std;

882 struct s

883 {

884 int distance;

885 int start;

886 int ending;

887 };

888 struct fn

889 {

890 bool operator()(s const&a,s const &b)

891 {

892 if(a.distance!=b.distance)

893 {

894 return a.distance<b.distance;

895 }

896 return a.start<b.start;

897 }

898 };

899 int main()

900 {

901 int n;

902 cin >> n;

903 int mat[n] = {0};

904 priority\_queue<s,vector<s>,fn> q;

905 struct s temp;

906 int l=1,r=n,coun=1;

907 q.push({n,l,r});

908 while(!q.empty())

909 {

910 temp = q.top();

911 q.pop();

912 int left = temp.start;

913 int right = temp.ending;

914 int mid = (left+right)/2;

915 if(temp.distance>0)

916 {

917 if(right>mid)

918 {

919 q.push({right-mid,mid+1,right}); // first push right child then left

920 }

921 if(left<mid)

922 {

923 q.push({mid-left,left,mid-1}); 924 }

925 }

926 mat[mid-1] = coun;

927 coun++;

928 for(int i=0; i<n; i++)

929 {

930 if(mat[i]==0)

931 {

932 cout << "\_" << " ";

933 }

934 else

935 {

936 cout << "X" << " ";

937 }

938

939 }

940 cout << endl;

941 }

942

943 }

944

945

946

947 # Rare elements

948

949 #include<iostream>

950 using namespace std;

951

952 struct node {

953 int x;

954 int y;

955 int level;

956 };

957

958 node q[1000];

959 int front = 0, back = 0;

960

961 void init() {

962 front = back = 0;

963 }

964

965 void push(int x, int y, int level) {

966 q[back].x = x;

967 q[back].y = y;

968 q[back].level = level;

969 back++;

970 }

971 node pop() {

972 return q[front++];

973 }

974 bool empty() {

975 return (front == back);

976 }

977

978

979 int a[100][100];

980 int rare[4][2];

981 int c;

982 int n;

983

984 bool valid(int r, int c) {

985 return (r >= 0 && r < n && c >= 0 && c < n); 986 }

987

988 int vis[100][100];

989

990 int xx[] = { -1, 0, 1, 0};

991 int yy[] = {0, 1, 0, -1};

992

993 int bfs(int sx, int sy, int dx, int dy) {

994

995 push(sx, sy, 0);

996 vis[sx][sy] = 1;

997

998 while (!empty()) {

999

1000 node temp = pop();

1001 if (temp.x == dx && temp.y == dy) return temp.level; 1002

1003 for (int i = 0; i < 4; i++) {

1004

1005 int valx = temp.x + xx[i];

1006 int valy = temp.y + yy[i];

1007 int lvl = temp.level + 1;

1008

1009 if (valid(valx, valy)) {

1010 if (a[valx][valy] == 1 && vis[valx][valy] == 0) {

1011 push(valx, valy, lvl); 1012 vis[valx][valy] = 1;

1013 }

1014 }

1015 }

1016 }

1017

1018 }

1019

1020

1021 int main() {

1022

1023

1024 int t; cin >> t;

1025 while (t--) {

1026 cin >> n;

1027 cin >> c;

1028

1029 init();

1030

1031 for (int i = 0; i < c; i++) {

1032 int x, y; cin >> x >> y;

1033

1034 x--; y--;

1035 rare[i][0] = x;

1036 rare[i][1] = y;

1037 }

1038

1039 for (int i = 0; i < n; i++) {

1040 for (int j = 0; j < n; j++) { 1041 cin >> a[i][j];

1042 }

1043 }

1044

1045 int ans = 10000;

1046

1047 for (int i = 0; i < n; i++) {

1048 for (int j = 0; j < n; j++) { 1049 int temp;

1050

1051 if (a[i][j] == 1) {

1052 temp = 0;

1053

1054 for (int k = 0; k < c; k++) { 1055

1056 init();

1057 for (int l = 0; l < 100; l++) 1058 for (int m = 0; m < 100; m++) 1059 vis[l][m] = 0; 1060

1061 int res = bfs(i, j, rare[k][0], rare[k][1]);

1062 temp = max(res, temp); 1063 }

1064

1065 ans = min(ans, temp); 1066 }

1067

1068 }

1069 }

1070 cout << ans << endl;

1071 }

1072

1073 return 0;

1074 }

1075

1076

1077

1078 # Sum binary tree

1079

1080 class Node

1081 {

1082 public:

1083 int data;

1084 Node \*left , \*right;

1085 };

1086

1087 Node \*newNode(int data)

1088 {

1089 Node \*temp = new Node;

1090 temp -> data = data;

1091 temp -> left = NULL;

1092 temp -> right = NULL;

1093

1094 return temp;

1095 }

1096

1097 int toSumTree(Node \*root)

1098 {

1099 if (root == NULL) return 0;

1100

1101 int old\_val = root->data;

1102 root->data = toSumTree(root->left) + toSumTree(root->right);

1103

1104 return root->data + old\_val;

1105

1106 }

1107

1108 void printInorder(Node \*root)

1109 {

1110 if (root == NULL) return ;

1111 printInorder(root->left);

1112 cout << " " << root->data;

1113 printInorder(root->right);

1114 }

1115

1116

1117 int main()

1118 {

1119 Node \*root;

1120 root = newNode(10);

1121 root->left = newNode(-2);

1122 root->right = newNode(6);

1123 root->left->left = newNode(8);

1124 root->left->right = newNode(-4);

1125 root->right->left = newNode(7);

1126 root->right->right = newNode(5);

1127

1128 toSumTree(root);

1129

1130 cout << "Inorder Traversal of the resultant tree is: \n";

1131 printInorder(root);

1132 return 0;

1133

1134 }

1135

1136

1137

1138

1139 # Day from a week

1140

1141 const string str[] = {"SUN", "MON", "TUE", "WED", "THU", "FRI", "SAT"};

1142 int dayofweek(int d, int m, int y)

1143 {

1144 int t[] = {11, 12, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}; 1145

1146 for (int i = 0; i < 12; i++)

1147 {

1148 double dd = (2.6 \* t[i] - 0.2) ; 1149 cout << dd << ' ';

1150 }

1151

1152 for (int i = 0; i < 12; i++)

1153 cout << t[i] << ' ';

1154 cout << endl;

1155

1156 //int t[] = {0, 3, 2, 5, 0, 5, 5, 1, 4, 6, 2, 4}; 1157 y -= m < 3;

1158 return (y + y / 4 + y / 400 - y / 100 + t[m - 1] + d) % 7;

1159 }

1160 int main()

1161 {

1162 while (1) {

1163 int d, m, y;

1164 cin >> d >> m >> y;

1165 int num = dayofweek(d, m, y);

1166 cout << str[num] << endl;

1167 break;

1168 }

1169 return 0;

1170 }

1171

1172 /\*

1173

1174 So consider this as an array : int t[] =

{11,12,1,2,3,4,5,6,7,8,9,10};

1175

1176 Now for all elements in array just do : (2.6\*m - 0.2) mod 7 parse

1177 the result as integer and you will get this: 0 3 2 5 0 3 5 1 4 6 2 4

1178

1179 \*/

1180

1181

1182

1183 # Largest bst

1184

1185 /\*

1186 Time Complexity: O(N)

1187 Space Complexity: O(N)

1188

1189 where 'N' is the total number of nodes in the binary tree.

1190 \*/

1191

1192 #include<bits/stdc++.h>

1193 using namespace std;

1194

1195 struct Node

1196 {

1197 int data;

1198 struct Node\* left;

1199 struct Node\* right;

1200 Node(int data)

1201 {

1202 this->data = data;

1203 this->left = NULL;

1204 this->right = NULL;

1205 }

1206 };

1207

1208 struct info

1209 {

1210 bool isValid;

1211 int size, min, max;

1212 };

1213

1214 info maxSize(Node\* currNode, int &maxBST)

1215 {

1216 if (currNode == NULL)

1217 {

1218 // isValid, size, min, max.

1219 return {true, 0, INT\_MAX, INT\_MIN}; 1220 }

1221

1222

1223 // Information of left and right subtrees. 1224 info left = maxSize(currNode -> left, maxBST); 1225 info right = maxSize(currNode -> right, maxBST); 1226

1227

1228 info currInfo;

1229

1230 currInfo.size = left.size + right.size + 1; 1231

1232

1233 currInfo.isValid = left.isValid & right.isValid; 1234

1235 // Current subtree must form a BST.

1236 currInfo.isValid &= (currNode -> data > left.max); 1237 currInfo.isValid &= (currNode -> data < right.min); 1238

1239 // Updating min and max for current subtree. 1240 currInfo.min = min(min(left.min, right.min), currNode -> data);

1241 currInfo.max = max(max(left.max, right.max), currNode -> data);

1242

1243

1244 if (currInfo.isValid == true)

1245 {

1246 maxBST = max(maxBST, currInfo.size); 1247 }

1248

1249 return currInfo;

1250 }

1251

1252

1253

1254 int main()

1255 {

1256 Node \*root = new Node(60);

1257 root->left = new Node(65);

1258 root->right = new Node(70);

1259 root->left->left = new Node(50);

1260

1261 int ans = 0;

1262 maxSize(root , ans);

1263 printf(" Size of the largest BST is %d\n", ans); 1264 return 0;

1265

1266 }

1267

1268

1269

1270

1271 # Closest leaf

1272

1273

1274 class BinaryTreeNode {

1275 public :

1276 T data;

1277 BinaryTreeNode<T> \*left;

1278 BinaryTreeNode<T> \*right;

1279

1280 BinaryTreeNode(T data) {

1281 this -> data = data;

1282 left = NULL;

1283 right = NULL;

1284 }

1285

1286 };

1287

1288 const int INF = 1e6;

1289

1290 // Function that returns the distance of the closest leaf in the sub tree.

1291 int closestLeafNodeInSubtree(BinaryTreeNode<int> \*root) { 1292 if (root == NULL) {

1293 return INF;

1294 }

1295 if (root->left == NULL && root->right == NULL) { 1296 // Node is a leaf node.

1297 return 0;

1298 }

1299

1300 int distLeft = closestLeafNodeInSubtree(root->left); 1301 int distRight = closestLeafNodeInSubtree(root->right); 1302

1303 return 1 + min(distLeft, distRight);

1304 }

1305

1306 // Helper function to calculate the closest leaf distance of the node.

1307 int

findClosestLeafNodeDistanceHelper(vector<BinaryTreeNode<int> \*> &ancestors,

1308 BinaryTreeNode<int> \*root, int x) {

1309

1310 if (root == NULL) {

1311 return INF;

1312 }

1313

1314 // If the required node is found, calculate the distance of the closest leaf node.

1315 if (root->data == x) {

1316 int result = closestLeafNodeInSubtree(root); 1317 int n = ancestors.size();

1318 for (int i = n - 1; i >= 0; --i) {

1319 int dist = n - i +

closestLeafNodeInSubtree(ancestors[i]);

1320 result = min(result, dist);

1321 }

1322 return result;

1323 }

1324 // Else check for other nodes by adding and removing

the nodes as ancestor and recur further.

1325 ancestors.push\_back(root);

1326

1327 int distLeft =

findClosestLeafNodeDistanceHelper(ancestors, root->left, x); 1328 int distRight =

findClosestLeafNodeDistanceHelper(ancestors, root->right, x);

1329

1330 ancestors.pop\_back();

1331

1332 return min(distLeft, distRight);

1333 }

1334

1335 int findClosestLeafNodeDistance(BinaryTreeNode<int> \*root, int x) {

1336 // Vector to store the ancestors of the node in the tree.

1337 vector<BinaryTreeNode<int> \*> ancestors; 1338

1339 return findClosestLeafNodeDistanceHelper(ancestors, root, x);

1340 }

1341

1342

1343

1344

1345

1346 # All posible BST

1347

1348 // For 0th ans = 1;

1349 void catalan(int n)

1350 {

1351 cpp\_int cat\_ = 1;

1352 cout << cat\_ << " "; // C(0)

1353 for (cpp\_int i = 1; i <=n; i++)

1354 {

1355 cat\_ \*= (4 \* i - 2);

1356 cat\_ /= (i + 1);

1357 cout << cat\_ << " ";

1358 }

1359 }

1360

1361

1362

1363 # Crow Pot

1364

1365 int minCrowPotStone()

1366 {

1367 int tot\_stone = 0;

1368 int temp[n + 9];

1369 temp[0] = a[0];

1370 for (int i = 1; i < n ; i++)

1371 temp[i] = a[i] - a[i - 1];

1372 for (int i = 0; i < m; i++)

1373 tot\_stone += (temp[i] \* (n - i) );

1374 return tot\_stone;

1375 }

1376 int main()

1377 {

1378 cin >> n >> m;

1379 for (int i = 0; i < n; i++)

1380 cin >> a[i];

1381 sort();

1382 cout << minCrowPotStone() << endl;

1383 }

1384

1385

1386

1387 # oil mine

1388

1389 int n , m , ans = INT\_MAX ;

1390 void calculateTotal(int i, int curr, int oil[], int visited[], int minV, int maxV, int comNum)

1391 {

1392 if (visited[i]) {

1393 int newMin = min(curr, minV);

1394 int newMax = max(curr, maxV);

1395

1396 if (comNum == n - 1) {

1397 ans = min(ans, newMax - newMin); 1398 }

1399 return;

1400 }

1401 visited[i] = 1;

1402 int j = (i + 1) % m;

1403

1404 calculateTotal(j, curr + oil[i] , oil, visited, minV, maxV, comNum);

1405

1406 int newMin = min(curr, minV);

1407 int newMax = max(curr, maxV);

1408

1409 calculateTotal(j , oil[i], oil, visited, newMin, newMax, comNum + 1 );

1410

1411 visited[i] = 0;

1412 return;

1413

1414 }

1415 int main()

1416 {

1417 cin >> n >> m;

1418 int oil[m] , visited[m];

1419 for (int i = 0; i < m; i++)

1420 cin >> oil[i] , visited[i] = 0;

1421

1422 if (n > m)

1423 {

1424 cout << -1 << endl;

1425 return 0;

1426 }

1427

1428

1429 for (int i = 0; i < m; i++)

1430 calculateTotal(i , 0 , oil , visited , INT\_MAX , INT\_MIN , 0);

1431 cout << ans << endl;

1432

1433 }

1434

1435

1436

1437 # Jewel Maze

1438

1439

1440

1441 /\*

1442 There is a maze that has one entrance and one exit. Jewels are placed in passages of the maze. You want to pick up the jewels after getting into the maze through the entrance and before getting out of it through the exit. You want to get as many jewels as possible, but you don’t want to take the same passage you used once.

1443

1444 When locations of a maze and jewels are given, find out the greatest number of jewels you can get without taking the same passage twice, and the path taken in this case. 1445

1446 Input

1447 There can be more than one test case in the input file. The first line has T, the number of test cases. Then the totally T test cases are provided in the following lines (T ≤ 10 ).

1448

1449 In each test case, In the first line, the size of the maze N (1 ≤ N ≤ 10) is given. The maze is N×N square-shaped. From the second line through N lines, information of the maze is given. “0” means a passage, “1” means a wall, and “2” means a location of a jewel. The entrance is located on the upper-most left passage and the exit is located on the lower-most right passage. There is no case where the path from the entrance to the exit doesn’t exist.

1450

1451 Output

1452 From the first line through N lines, mark the path with 3 and output it. In N+1 line, output the greatest number of jewels that can be picked up. Each test case must be output separately as a empty.

1453

1454 MAX DIAMONDS COLLECTED AND ITS PATH IS THE OUTPUT. 1455

1456 \*/

1457

1458 #include<iostream>

1459 using namespace std;

1460

1461 int n;

1462 int a[100][100];

1463

1464 int dx[] = {-1,0,1,0};

1465 int dy[] = {0,1,0,-1};

1466

1467 bool valid(int x, int y){

1468 return ((a[x][y] == 0 || a[x][y] == 2) && x>=0 && x<n && y>=0 && y<n);

1469 }

1470

1471 int ans[50][50];

1472 //int paths;

1473 int value = -100;

1474

1475 void print(){

1476 for(int i = 0; i<n;i++){

1477 for(int j = 0; j<n; j++){

1478 cout<<ans[i][j]<<" ";

1479 }

1480 cout<<endl;

1481 }

1482 cout<<endl;

1483 }

1484

1485 void solve(int r, int c, int diamonds){

1486

1487 if(r == n-1 && c == n-1){

1488 if(diamonds>value){

1489 value = diamonds;

1490 for(int i = 0; i<n; i++){

1491 for(int j = 0; j<n; j++){

1492 ans[i][j] = a[i][j];

1493 //print();

1494 }

1495 }

1496 }

1497 }

1498

1499 for(int i=0; i<4; i++){

1500

1501 int x = r + dx[i];

1502 int y = c + dy[i];

1503

1504 if(valid(x,y)){

1505

1506 int check = (a[x][y] == 2) ? 1:0; 1507 a[x][y] = 3;

1508 solve(x,y,diamonds + check);

1509 a[x][y] = (check == 1) ? 2:0;

1510 }

1511 }

1512 }

1513

1514

1515 int main(){

1516

1517 cin>>n;

1518 for(int i =0; i<n; i++) 1519 for(int j =0; j<n; j++) 1520 cin>>a[i][j];

1521

1522 /\* here 2 is diamond 1523 0 means a passage 1524 1 means a wall

1525 \*/

1526 //paths = 0;

1527 value = -100;

1528 a[0][0] = 3;

1529 solve(0,0,0);

1530 cout<<value<<endl; 1531 print();

1532

1533 return 0;

1534 }

1535

1536

1537

1538

1539

1540

1541

1542

1543

1544

1545 # Lanching Bomb

1546

1547

1548 int dx[] = { -1, 0, 1, 0}; 1549 int dy[] = {0, 1, 0, -1}; 1550

1551

1552 void solve() {

1553

1554 if (a[sx][sy] == 0)return; 1555

1556 vis[sx][sy] = true; 1557 push(sx, sy, 1);

1558

1559

1560 while (!empty()) { 1561

1562 node temp = pop(); 1563

1564 int x, y, l;

1565 x = temp.x;

1566 y = temp.y;

1567 l = temp.l;

1568

1569

1570 ans = max(ans, l); 1571

1572

1573 for (int i = 0; i < 4; i++) {

1574

1575 int xx = x + dx[i];

1576 int yy = y + dy[i];

1577

1578 if (valid(xx, yy) && a[xx][yy] == 1) { 1579 vis[xx][yy] = true;

1580 push(xx, yy, l + 1);

1581 }

1582

1583 }

1584

1585 }

1586

1587 }

1588

1589 int main() {

1590

1591 int t; cin >> t;

1592 while (t--) {

1593

1594 /\* FIRST COLUMN IS TAKEN AS INPUT AND THEN ROW ACC. 1595 TO TEST CASES \*/

1596 cin >> m >> n;

1597

1598

1599 for (int i = 0; i < n; i++) {

1600 for (int j = 0; j < m; j++) {

1601 vis[i][j] = false;

1602 cin >> a[i][j];

1603 }

1604 }

1605 /\* acc to the test cases first col is taken as input and then row

1606 I have taken the input as row and then col so write it down as

1607 cin>>sy>>sx

1608 \*/

1609 cin >> sx >> sy;

1610 sx--; sy--; /\* zero based indexing \*/ 1611 init();

1612 ans = -1;

1613 solve();

1614 cout << ans << endl;

1615 }

1616

1617

1618 return 0;

1619 }

1620

1621

1622

1623 # Count BSt and BS

1624

1625 #include<iostream>

1626 using namespace std;

1627 #define ll long long

1628

1629 const ll MOD = 1e9 + 7;

1630 const ll N = 500;

1631 ll fact[1005] , n = 1000 ;

1632

1633

1634 ll normal(ll &a)

1635 {

1636 a %= MOD;

1637 if (a < 0) a += MOD;

1638 }

1639 ll ModMul(ll a, ll b)

1640 {

1641 normal(a), normal(b);

1642 return (a \* b) % MOD;

1643 }

1644 ll ModPow(ll b, ll p)

1645 {

1646 ll r = 1;

1647 while (p)

1648 {

1649 if (p & 1) r = ModMul(r , b); 1650 b = ModMul(b, b);

1651 p >>= 1;

1652 }

1653 return r;

1654 }

1655 ll modInverse(ll a) {

1656 return ModPow(a, MOD - 2);

1657 }

1658

1659 ll ModDiv(ll a, ll b)

1660 {

1661 return ModMul(a , ModPow(b , MOD - 2)); 1662 }

1663

1664 void func()

1665 {

1666 fact[0] = 1;

1667 for (int i = 1; i <= n; i++)

1668 fact[i] = ModMul(fact[i - 1] , i ); 1669 }

1670

1671 ll NcR(ll n, ll r)

1672 {

1673 if (n < r)

1674 return 0;

1675 if (r == 0)

1676 return 1;

1677

1678

1679 return (fact[n] \* modInverse(fact[r]) % MOD 1680 \* modInverse(fact[n - r]) % MOD) 1681 % MOD;

1682 }

1683

1684 long long NCR(ll n, ll r) { /// O(r)

1685 if (r > n - r) r = n - r; // because C(n, r) == C(n, n - r)

1686 long long ans = 1;

1687 for (int i = 1; i <= r; i++) {

1688 ans \*= n - r + i; /// or ans \*= n-i+1 1689 ans /= i;

1690 }

1691 return ans;

1692 }

1693

1694 unsigned long int catalan(unsigned long int n) 1695 {

1696 // Calculate value of 2nCn

1697 unsigned long int \_2nCn = NCR(2 \* n, n); 1698

1699 // return 2nCn/(n+1)

1700 return \_2nCn / (n + 1);

1701 }

1702

1703 unsigned long int countBST(unsigned int n)

1704 {

1705 unsigned long int count = catalan(n);

1706 // return nth catalan number

1707 return count;

1708 }

1709

1710 unsigned long int countBT(unsigned int n)

1711 {

1712 unsigned long int count = catalan(n);

1713 // return count \* n!

1714 return count \* fact[n];

1715 }

1716

1717

1718 int main()

1719 {

1720 ll n , r , t;

1721

1722 func();

1723

1724

1725 int count1, count2, nn = 5;

1726

1727 // find count of BST and binary trees with n nodes 1728 count1 = countBST(nn);

1729 count2 = countBT(nn);

1730

1731 // print count of BST and binary trees with n nodes 1732 cout << "Count of BST with " << nn << " nodes is " << count1 << endl;

1733 cout << "Count of binary trees with " << nn << " nodes is " << count2;

1734 }

1735

1736

1737

1738

1739

1740 # Frog Jump

1741

1742 #include<iostream>

1743 using namespace std;

1744

1745 int n, m;

1746 int a[100][100];

1747

1748 int r[] = {0, -1, 0, 1};

1749 int c[] = { -1, 0, 1, 0};

1750

1751 int dp[100][100];

1752

1753 bool valid(int x, int y) {

1754 return (x > 0 && x <= n && y > 0 && y <= m && a[x][y] == 1);

1755 }

1756

1757 void solve(int sx, int sy, int dx, int dy, int ans) { 1758

1759 if (dp[sx][sy] > ans) {

1760 dp[sx][sy] = ans;

1761

1762 for (int i = 0; i < 4; i++) {

1763

1764 int x = sx + r[i];

1765 int y = sy + c[i];

1766

1767 if (valid(x, y)) {

1768 int temp;

1769 if (y == sy)temp = 1;

1770 if (x == sx)temp = 0;

1771 solve(x, y, dx, dy, ans + temp); 1772 }

1773 }

1774 }

1775

1776 }

1777

1778 int main() {

1779

1780 cin >> n >> m;

1781 for (int i = 1; i <= n; i++) {

1782 for (int j = 1; j <= m; j++) {

1783 dp[i][j] = 1000000;

1784 cin >> a[i][j];

1785 }

1786 }

1787

1788 int sx, sy, dx, dy;

1789 cin >> sx >> sy >> dx >> dy;

1790

1791 solve(sx, sy, dx, dy, 0);

1792

1793 cout << dp[dx][dy] << endl;

1794

1795 return 0;

1796 }

1797

1798

1799

1800

1801

1802 # JOb Scheduling

1803

1804

1805 struct node

1806 {

1807 int start , finish , profit;

1808 };

1809

1810 bool cmp(node a, node b)

1811 {

1812 return a.finish < b.finish;

1813 }

1814

1815 int latestconflict(node arr[], int idx, int n) 1816 {

1817 int low = 0 , high = idx - 1 , j = -1; 1818 while (low <= high)

1819 {

1820 int mid = (low + high) / 2;

1821 if (arr[mid].finish <= arr[idx].start) 1822 {

1823 j = mid;

1824 low = mid + 1;

1825 }

1826 else high = mid - 1;

1827 }

1828 return j;

1829 }

1830

1831 void solve(node arr[], int n)

1832 {

1833 int table[n + 9];

1834 for (int i = 0; i < n + 9; i++)

1835 table[i] = 0;

1836

1837 int ans = 0 ;

1838 table[0] = arr[0].profit;

1839 for (int i = 1; i < n; i++)

1840 {

1841 int l = latestconflict(arr, i, n); 1842 int res = 0 ;

1843 if (l != -1) res = table[l] + arr[i].profit; 1844

1845 table[i] = max(table[i - 1] , res); 1846 }

1847

1848 cout << table[n - 1] << endl;

1849 }

1850

1851

1852 int main()

1853 {

1854 int n;

1855 cin >> n ;

1856 node arr[n + 9];

1857 for (int i = 0; i < n; i++)

1858 {

1859 cin >> arr[i].start >> arr[i].finish >> arr[i].profit;

1860 }

1861 sort(arr , arr + n , cmp );

1862

1863 solve(arr , n);

1864

1865 }

1866

1867

1868

1869

1870 # Max pipe

1871

1872

1873 int dp[1000][1000];

1874 int arr[1000];

1875

1876 int solve(int pos , int curr)

1877 {

1878 if (pos == n)

1879 {

1880 int tmp = abs( tot - curr - curr); 1881 if (tmp < ans)

1882 {

1883 ans = tmp;

1884 counT = max(curr , tot - curr); 1885 }

1886 return 1;

1887 }

1888

1889 if (dp[pos][curr] != -1)

1890 return dp[pos][curr];

1891

1892 dp[pos][curr] = solve(pos + 1 , curr + arr[pos]) + solve(pos + 1 , curr);

1893 return dp[pos][curr];

1894 }

1895

1896 int main()

1897 {

1898 cin >> n ;

1899 for (int i = 0; i < n; i++)

1900 cin >> arr[i] , tot += arr[i];

1901

1902 for (int i = 0; i <= n; i++)

1903 {

1904 for (int j = 0; j <= tot; j++)

1905 {

1906 dp[i][j] = -1;

1907 }

1908 }

1909 ans = 100000;

1910 solve(0, 0) ;

1911 cout << counT << endl;

1912 }

1913

1914

1915

1916

1917

1918 # Max Grid Sum

1919

1920

1921 int main()

1922 {

1923 int grid[N][M] = { { 10, 10, 2, 0, 20, 4 }, 1924 { 1, 0, 0, 30, 2, 5 },

1925 { 0, 10, 4, 0, 2, 0 },

1926 { 1, 0, 2, 20, 0, 4 }

1927 };

1928

1929 int n = N , m = M;

1930

1931 for (int i = 1; i < n; i++) {

1932 grid[0][i] += grid[0][i - 1];

1933 }

1934

1935 for (int i = 1; i < m; i++) {

1936 grid[i][0] += grid[i - 1][0];

1937

1938 for (int j = 1; j < n; j++) {

1939 grid[i][j] += max(grid[i][j - 1], grid[i - 1][j]);

1940 }

1941 }

1942

1943 cout << grid[M - 1][N - 1];

1944

1945 }

1946

1947

1948

1949

1950 # Rotate Image

1951

1952 class Solution {

1953 public:

1954 void rotate(vector<vector<int>>& matrix) { 1955 transpose(matrix);

1956 reflect(matrix);

1957 }

1958

1959 void transpose(vector<vector<int>>& matrix) 1960 {

1961 int n = matrix.size();

1962 for (int i = 0; i < n; i++)

1963 {

1964 for (int j = i + 1; j < n; j++) 1965 {

1966 swap(matrix[i][j] , matrix[j][i]); 1967 }

1968 }

1969 }

1970

1971 void reflect(vector<vector<int>>& matrix) 1972 {

1973 int n = matrix.size();

1974 for (int i = 0; i < n; i++)

1975 {

1976 for (int j = 0; j < n / 2; j++) 1977 {

1978 swap(matrix[i][j] , matrix[i][n - j - 1]); 1979 }

1980 }

1981 }

1982 };

1983

1984

1985

1986

1987

1988 # DOctor Probability

1989

1990 /\*

1991 https://www.geeksforgeeks.org/samsung-interview-experience-s et-39-campus-r-d-noida/

1992 https://www.careercup.com/page?pid=samsung-interview-questio ns

1993 A Doctor travels from a division to other division where divisions are connected like a graph(directed graph) and 1994 the edge weights are the probabilities of the doctor going from that division to other connected division but the 1995 doctor stays 10mins at each division now there will be given time and had to find the division in which he will be 1996 staying by that time and is determined by finding division which has high probability.

1997 Input is number of test cases followed by the number of nodes, edges, time after which we need to find the division 1998 in which he will be there, the edges starting point, end point, probability.

1999 Note: If he reaches a point where there are no further nodes then he leaves the lab after 10 mins and the traveling 2000 time is not considered and during that 10min at 10th min he will be in next division, so be careful

2001 2

2002 6 10 40

2003 1 2 0.3 1 3 0.7 3 3 0.2 3 4 0.8 2 4 1 4 5 0.9 4 4 0.1 5 6 1.0 6 3 0.5 6 6 0.5

2004 6 10 10

2005 1 2 0.3 1 3 0.7 3 3 0.2 3 4 0.8 2 4 1 4 5 0.9 4 4 0.1 5 6 1.0 6 3 0.5 6 6 0.5

2006 6 0.774000

2007 3 0.700000

2008 \*/

2009

2010 #include<iostream>

2011 using namespace std;

2012

2013 void docProb(double \*\*graph, int nodes, int time, int curNode, double p, double \*answer){

2014 if(time <= 0){

2015 answer[curNode] += p;

2016 return;

2017 }

2018

2019 for(int i=1; i<=nodes; i++){

2020 if(graph[curNode][i] != 0){

2021 p \*= graph[curNode][i];

2022 docProb(graph, nodes, time - 10, i, p, answer); 2023 p /= graph[curNode][i];

2024 }

2025 }

2026

2027 }

2028

2029 int main(){

2030 int t;

2031 cin >> t;

2032 while(t--){

2033 int nodes, edges, time;

2034 cin >> nodes >> edges >> time;

2035

2036 double \*\*arr = new double\*[nodes];

2037 for(int i=1; i<=nodes; i++){

2038 arr[i] = new double[nodes];

2039 for(int j=1; j<=nodes; j++){

2040 arr[i][j] = 0;

2041 }

2042 }

2043

2044 int from, to;

2045 double prob;

2046 for(int i=0; i<edges; i++){

2047 cin >> from >> to >> prob;

2048 arr[from][to] = prob;

2049 }

2050

2051 /\* Initalise answer and function call \*/ 2052 double answer[nodes] = {0.0};

2053 docProb(arr, nodes, time, 1, 1.0, answer); 2054

2055 /\* Select max Probability node \*/ 2056 double finalProb = 0.0;

2057 int finalDivison = 0;

2058

2059 for(int i=1; i<=nodes; i++){

2060 if(answer[i] > finalProb){

2061 finalProb = answer[i];

2062 finalDivison = i;

2063 }

2064 }

2065 cout << finalDivison << " " << finalProb << "\n"; 2066 }

2067 return 0;

2068 }

2069

2070

2071

2072

2073

2074

2075 /// Floyd Warshall

2076 for(int k=0;k<nodes;k++){

2077 for(int i=0;i<nodes;i++){ 2078 for(int j=0;j<nodes;j++){ 2079 if(i==k||j==k)

2080 continue;

2081

cost[i][j]=min(cost[i][j],cost[i][k]+cost[k][j]); 2082 }

2083 }

2084 }

2085

2086

2087

2088

2089 # Convex Hull

2090

2091 /\*

2092 Given random points in a 2-D plane, construct a convex polygon with minimum area of covering and

2093 which encompasses all the given points.

2094 \*/

2095 #include<bits/stdc++.h>

2096 int cou = 0;

2097

2098 struct Point{

2099 int x, y;

2100 };

2101

2102 int orientation(Point p, Point q, Point r){ 2103 int val = (q.y - p.y) \* (r.x - q.x) - 2104 (q.x - p.x) \* (r.y - q.y);

2105

2106 if (val == 0) return 0;

2107 return (val > 0)? 1: 2;

2108 }

2109

2110 bool cmp(Point &a, Point &b){

2111 if(a.x==b.x&&a.y==b.y)

2112 cou++;

2113

2114 if(a.x == b.x)

2115 return a.y < b.y;

2116 else

2117 return a.x < b.x;

2118 }

2119

2120 bool myFunc(Point &a, Point &b){

2121 return (a.x==b.x && a.y==b.y);

2122 }

2123

2124 void convexHull(Point \*points, int n){

2125 cou = 0;

2126 if (n < 3){

2127 cout << "-1";

2128 return;

2129 }

2130

2131 vector<Point> hull;

2132

2133 int l = 0;

2134 for (int i = 1; i < n; i++)

2135 if (points[i].x < points[l].x)

2136 l = i;

2137

2138 int p = l, q;

2139 do{

2140 hull.push\_back(points[p]);

2141

2142 q = (p+1)%n;

2143

2144 for (int i = 0; i < n; i++)

2145 {

2146 if (orientation(points[p], points[i], points[q]) == 2)

2147 q = i;

2148 }

2149 p = q;

2150

2151 } while (p != l);

2152

2153 sort(hull.begin(), hull.end(), cmp); 2154

2155 auto ip = unique(hull.begin(), hull.end(), myFunc); 2156

2157 hull.resize(std::distance(hull.begin(), ip)); 2158

2159 if(n < 4 && cou > 0 || hull.size() < 3){ 2160 cout << "-1";

2161 return;

2162 }

2163 else{

2164 for (int i = 0; i < hull.size(); i++){ 2165 if(i != hull.size() - 1)

2166 cout << hull[i].x << " " << hull[i].y << ", ";

2167 else

2168 cout << hull[i].x << " " << hull[i].y; 2169 }

2170 }

2171 }

2172

2173 int main(){

2174 int t, n;

2175 cin >> t;

2176 while(t--){

2177 cin >> n;

2178 Point \*points = new Point[n];

2179

2180 for(int i=0; i<n; i++){

2181 cin >> points[i].x >> points[i].y; 2182 }

2183

2184 convexHull(points, n);

2185 cout << "\n";

2186 }

2187 return 0;

2188 }

2189

2190

2191 ### Two Problem Mixed

2192

2193

2194 Given below are the raw materials quantities and their respective selling price(if sold as raw).

2195

2196 D --> No of CPUs

2197 E --> No of memory chips

2198 F --> No of boards

2199 d --> Selling price of CPU

2200 e --> Selling price of Memory chips

2201

2202 We are given N Computer configurations like below : 2203 Di, Ei, Fi, SPi, which are the CPU, Chips, Boards and one unit selling price for ith computer respectively.

2204 Our task is to maximize the final cost.

2205 Constraints:

2206 1. Can use at Max 3 different Configurations 2207 2. We can use 1 configuration multiple times 2208 3. Remaining Inventories can be sold on its selling price 2209

2210 Input:

2211 T --> Number of test cases.

2212 D E F d e --> Inventories

2213 N --> Total Configuration Count

2214 Di Ei Fi SPi

2215 ...

2216 Dn En Fn SPn

2217

2218 1<=T<=10

2219 1<= D, E, F <= 100

2220 1<= d, e <=100000

2221 1<=N<=8

2222

2223 Output:

2224 First Line print the Case #testCaseNumber

2225 Second Line Print Maximum Cost per test case in each line. 2226

2227 Sample Input:

2228 1 --> Total Test Case

2229 10 10 10 2 1 --> D E F d e

2230 1 --> PC Configuration Count

2231 1 2 2 3 --> D1 E1 F1 SP1

2232

2233 Sample Output:

2234 Case #1

2235 30

2236

2237

2238 Solution:

2239

2240 #include<iostream>

2241

2242 using namespace std;

2243

2244 #define rep(i,a,n) for(int i =a; i < n; i++) 2245 #define repe(i,a,n) for(int i =a; i <= n; i++) 2246

2247 int D,E,F,d,e;

2248 int config;

2249 int answer = 0;

2250

2251 struct configuration

2252 {

2253 int D,E,F,SPi;

2254 };

2255 configuration m[9];

2256

2257 void solve(int index, int counta, int D, int E, int F, int cost )

2258 {

2259

2260 if(index >= config || counta == 3)

2261 {

2262 cost += D\*d + E\*e;

2263 if(cost > answer)

2264 answer = cost;

2265 return;

2266 }

2267 solve(index + 1, counta, D,E,F,cost);

2268

2269 int i = 1;

2270

2271 while(true)

2272 {

2273 if( D - m[index].D\*i >= 0 && E - m[index].E\*i >=0 && F - m[index].F\*i >= 0 )

2274 {

2275 solve(index+1,counta+1,D- m[index].D \*i,E - m[index].E \*i,F- m[index].F\*i, cost+ m[index].SPi \* i); 2276 ++i;

2277 }

2278 else

2279 {

2280 break;

2281 }

2282 }

2283 return;

2284

2285 }

2286

2287 int main()

2288 {

2289 int t;

2290 cin >> t;

2291 repe(\_cases,1,t)

2292 {

2293

2294 answer = 0;

2295 cin >> D >> E >> F >> d >> e;

2296

2297 cin >> config;

2298

2299 rep(i,0,config)

2300 {

2301 cin >> m[i].D >> m[i].E >> m[i].F >> m[i].SPi; 2302 }

2303 solve(0,0,D,E,F,0);

2304 cout << "Case #"<<\_cases << "\n" << answer <<"\n"; 2305

2306 }

2307

2308 return 0;

2309 }

2310 ------------------------------------------------------------ ---------

2311

2312

2313

2314 You want to cut a piece of paper by a certain fixed rule to make some pieces of white or

2315 blue colored square paper with various sizes. 2316

2317 If the size of the entire paper is N×N (N = 2^K; 1 <= K <= 7; K = natural number), the cutting rules

2318 are as below.

2319

2320 ‘If the entire piece of paper is not colored the same, cut the middle part horizontally and vertically

2321 to divide it into the same sized four pieces of paper,

2322 (N/2)×(N/2), as with I, II, III, IV in < FIG. 2 >. 2323

2324 For each I, II, III and IV, cut and divide again in the same way if one entire piece of paper

2325 is not colored the same, and make them into the same sized four pieces of paper. Continue until each and

2326 every piece of paper has only one color of white or blue.’ 2327

2328 When you finish, < FIG. 3 > shows the first division of < FIG. 1 > and < FIG. 4 >

2329 shows the final version of 9 pieces of white paper and 7 pieces of blue paper of various sizes.

2330

2331 If the length of an edge of the first given piece of paper, N, and

2332 the color information (white or blue) inside each square are given, create a calculation program

2333 that assesses how many white/blue pieces of paper are. 2334

2335 Time limit: 1 second (java: 2 seconds)

2336

2337 [Input]

2338

2339 Input may include many test cases. The number of test cases, T, is given on the first line of input and then the amount of T of test cases is given in a line. (T <= 30)

2340 The length of an edge of the first given piece of paper, N, is given for the first line of each test case.

2341 From the next line through to the amount of N lines, the color information is given separately as blanks. 0 indicates white and 1 indicates blue.

2342

2343 [Output]

2344

2345 For each test case, you should print "Case #T" in the first line where T means the case number.

2346

2347 For each test case, you should output the number of white pieces of paper and blue pieces of paper separately as blanks on the first line of each test case.

2348

2349 [I/O Example]

2350 Input

2351 2

2352 8

2353 1 1 0 0 0 0 1 1

2354 1 1 0 0 0 0 1 1

2355 0 0 0 0 1 1 0 0

2356 0 0 0 0 1 1 0 0

2357 1 0 0 0 1 1 1 1

2358 0 1 0 0 1 1 1 1

2359 0 0 1 1 1 1 1 1

2360 0 0 1 1 1 1 1 1

2361

2362

2363 16

2364 1 0 0 1 0 0 0 0 0 0 1 1 0 1 1 1

2365 1 1 0 1 0 1 1 0 0 0 0 0 0 0 0 0

2366 0 0 0 0 1 0 1 1 1 1 0 0 1 0 0 1

2367 1 1 0 0 1 0 0 1 0 0 1 0 1 1 1 0

2368 0 1 1 1 0 0 1 1 0 0 1 0 0 1 1 1

2369 1 0 1 1 0 0 0 1 0 1 0 1 0 0 1 1

2370 1 1 1 1 1 1 0 0 1 1 1 1 1 0 0 0

2371 1 1 0 1 0 1 0 0 1 0 1 1 1 0 0 1

2372 1 1 1 1 1 1 0 0 1 0 1 1 0 1 1 0

2373 1 0 0 1 1 1 0 0 0 0 1 1 1 1 0 0

2374 1 0 0 1 1 1 1 0 0 0 1 1 0 1 0 1

2375 1 1 1 0 1 1 0 0 1 1 1 1 1 1 0 1

2376 1 1 1 1 1 1 0 0 0 0 1 1 1 1 0 0

2377 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0 0

2378 1 1 0 0 0 0 0 0 1 1 0 1 1 0 0 0

2379 1 1 0 0 1 1 0 0 0 1 1 1 1 0 0 0

2380

2381

2382

2383 Output

2384

2385 Case #1

2386 9 7

2387

2388 Case #2

2389 88 99

2390 Solution :

2391 #include <iostream>

2392 #include <cstdio>

2393 #include <cstring>

2394 using namespace std;

2395 #define debug(x) cout << '>' << #x << ':' << x << endl; 2396 const int maxn = 129;

2397 int white = 0, blue = 0;

2398 bool checkSame(bool arr[maxn][maxn], int sti, int stj, int size)

2399 {

2400 bool color = arr[sti][stj];

2401 for(int i = sti; i < sti + size; i++){

2402 for(int j = stj; j < stj + size; j++){ 2403 if(arr[i][j] != color){

2404 return false;

2405 }

2406 }

2407 }

2408 return true;

2409 }

2410 void solve(bool arr[maxn][maxn], int size, int sti, int stj) 2411 {

2412 bool same = checkSame(arr, sti, stj, size); 2413

2414 if(!same){

2415 solve(arr, size / 2, sti, stj);

2416 solve(arr, size / 2, sti + size/2, stj); 2417 solve(arr, size / 2, sti, stj + size/2); 2418 solve(arr, size / 2, sti + size/2, stj + size/2);

2419 }

2420 else{

2421 (arr[sti][stj]) ? ++blue : ++white ; 2422 }

2423 }

2424 int main()

2425 {

2426 int test ;

2427 cin >> test ;

2428 for(int l = 1; l <= test; l++){ 2429 white = 0;

2430 blue = 0;

2431 int size ;

2432 cin >> size;

2433 bool arr[maxn][maxn];

2434 for(int i = 0; i < size; i++){ 2435 for(int j = 0; j < size; j++){ 2436 cin >> arr[i][j] ; 2437 }

2438 }

2439 solve(arr, size, 0, 0);

2440 cout << "Case #" << l << endl; 2441 cout << white << " " << blue << endl; 2442 }

2443 return 0;

2444 }

2445

2446

2447

2448

2449

2450

2451

2452

2453

2454

2455

2456

2457

2458

2459

2460

2461

2462