```cpp
/*
 * FILE: {{untitled}}
 *
 * @author: Arafat Hasan Jenin <arafathasanjenin[at]gmail[dot]com>
 *
 * LINK:
 *
 * DATE CREATED: {{long_date}}
 * LAST MODIFIED: __last_modified
 *
 * DESCRIPTION:
 *
 * DEVELOPMENT HISTORY:
 * Date         Version     Description
 * ---------------------------------------------------------------------
 * {{short_date}}     1.0         {{File Created}}
 *
 *                 _/  _/_/_/_/  _/      _/  _/_/_/  _/      _/
 *                _/  _/        _/_/    _/    _/    _/_/    _/
 *               _/  _/_/_/    _/  _/  _/    _/    _/  _/  _/
 *        _/    _/  _/        _/    _/_/    _/    _/    _/_/
 *         _/_/    _/_/_/_/  _/      _/  _/_/_/  _/      _/
 */

//////////////////////////////////////////////////////////////////////

#include <iostream>
#include <climits>
#include <cmath>
#include <cstring>
#include <cctype>
#include <cstdio>
#include <cstdlib>
#include <iomanip>
#include <utility>
#include <sstream>
#include <algorithm>
#include <stack>
#include <set>
#include <list>
#include <map>
#include <unordered_map>
#include <queue>
#include <deque>
#include <vector>
#include <tuple>
#include <stdint.h> //uint32_t
#include <functional>
#include <bitset>

using namespace std;

typedef long long           ll;
typedef double              lf;
typedef unsigned long long  ull;
typedef pair<int, int>      pii;
typedef vector<pii>         vpii;
```

```
58  typedef vector<int>         vi;
59
60  #define _USE_MATH_DEFINES
61
62  #define FileIn(file)        freopen("input.txt", "r", stdin)
63  #define FileOut(file)       freopen("output.txt", "w", stdout)
64  #define __FastIO            ios_base::sync_with_stdio(false); cin.tie(0);
    cout.tie(0)
65
66  #define forr(i, a, b)   for (__typeof (a) i=a; i<=b; i++)
67  #define rof(i, b, a)    for (__typeof (a) i=b; i>=a; i--)
68  #define rep(i, n)       for (__typeof (n) i=0; i<n; i++)
69  #define forit(i, s)     for (__typeof ((s).end ()) i = (s).begin (); i !=
    (s).end (); ++i)
70  #define all(ar)         ar.begin(), ar.end()
71  #define fill(a, val)    memset((a), (val), sizeof((a)))
72  #define clr(a)          memset((a), 0, sizeof((a)))
73  #define sz(a)           (int) a.size()
74
75  #define sfll(a)         scanf("%lld", &a)
76  #define pfll(a)         printf("%lld", a)
77  #define sflf(a)         scanf("%lf", &a)
78  #define pflf(a)         printf("%lf", a)
79  #define sff(a)          scanf("%f", &a)
80  #define pff(a)          printf("%f", a)
81  #define sf(a)           scanf("%d", &a)
82  #define pf(a)           printf("%d", a)
83
84  #define pb              push_back
85  #define gc              getchar
86  #define eb              emplace_back
87
88  #ifndef ONLINE_JUDGE
89      #define sp              cerr << ' '
90      #define nl              cerr << '\n'
91      #define ckk             cerr << "###############\n"
92      #define debug1(x)       cerr << #x << ": " << x << endl
93      #define debug2(x, y)    cerr << #x << ": " << x << '\t' << #y << ": " << y
        << endl
94      #define debug3(x, y, z) cerr << #x << ": " << x << '\t' << #y << ": " << y
        << '\t' << #z << ": " << z << endl
95  #else
96      #define sp
97      #define nl
98      #define ckk
99      #define debug1(x)
100     #define debug2(x, y)
101     #define debug3(x, y, z)
102 #endif
103
104 #define max(a, b)       (a < b ? b : a)
105 #define min(a, b)       (a > b ? b : a)
106 #define sq(a)           (a * a)
107
108 /////////////// BIT SET /////////////
109
110 // 2^n
```

```
112  // Check ith bit of integer n, 0 or 1
113  #define bitchk(n, i)        ((n & (1 << (i)))? 1 : 0)
114  //set ith bit ON of the integer n
115  #define bit_on(n, i)        n = (n | (1 << (i)))
116  //set ith bit OFF of the intger n
117  #define bit_off(n, i)       n = (n & ~(1 << (i)))
118  // Toggle ith bit of integer n, set 0 if 1, set 1 if 0
119  #define bit_toggle(n, i)    n = (n ^ ( 1 << (i)))
120  // Set ith bit to x (x is bool, 1 or 0) of the integer n
121  #define bit_setx(n, x, i)   n = (n ^ ((-(x) ^ n) & (1 << (i))))
122
123  /// int month[]={-1,31,28,31,30,31,30,31,31,30,31,30,31}; //Not Leap Year
124  /// int dx[]= {1,0,-1,0};int dy[]= {0,1,0,-1}; //4 Direction
125  /// int dx[]={1,1,0,-1,-1,-1,0,1};int dy[]={0,1,1,1,0,-1,-1,-1};//8 Direction
126  /// int dx[]= {2,1,-1,-2,-2,-1,1,2};int dy[]= {1,2,2,1,-1,-2,-2,-1}; //Knight
     Direction
127
128  #define PI              acos(-1.0)
129  #define INF             0x7fffffff
130  #define MOD             1000000007
131  #define EPS             1e-7
132  #define MAXN            32001
133  #define MAXS            100000008 // 1e8+8
134  #define MAX             10000007 // 1e7+7
135
136  template<class T>
137  void debug (T t) {
138      cout << t << endl;
139  }
140
141  template<typename T, typename ...Args>
142  void debug (T t, Args... args) {
143      cout << t << '\t';
144      debug (args...);
145  }
146
147  ////////////////////////////////////////////////////////////////
148  ////////////////////////// START HERE //////////////////////////
149  ////////////////////////////////////////////////////////////////
150
151
152  ////////////////////////// BFS //////////////////////////
153
154  class BFS_DIJKSTRA {
155          vector<int> adj [100]; // for BFS & bfs_vis
156          int cost[100]; // for BFS & bfs_vis
157          int visited [100]; //for bfs_vis
158
159          int bfs (int s, int n) {
160              int i, cn, v, sz;
161
162              for (i = 0; i < n; i++)
163                  cost[i] = INT_MAX;
164
165              queue<int> Q;
166              Q.push (s);
167              cost[s] = 0;
```

```
168
169             while (!Q.empty() ) {
170                 cn = Q.front();
171                 Q.pop();
172                 sz = (int) adj[cn].size();
173
174                 for (i = 0; i < sz; i++) {
175                     v = adj[cn][i];
176
177                     if (cost[cn] + 1 < cost[v]) {
178                         Q.push (v);
179                         cost[v] = cost[cn] + 1;
180                     }
181                 }
182             }
183
184             return 0;
185         }
186
187
188         void bfs_visited (int s, int n) {
189             int u, v, i, m;
190
191             for (i = 0; i < n; i++) {
192                 visited[i] = 0;
193                 cost[i] = -1;
194             }
195
196             queue<int> Q;
197             Q.push (s);
198             visited[s] = 1;
199             cost[s] = 0;
200
201             while (!Q.empty() ) {
202                 u = Q.front();
203                 Q.pop();
204                 m = (int) adj[u].size();
205
206                 for (i = 0; i < m; i++) {
207                     if (visited[adj[u][i]] == 0) {
208                         v = adj[u][i];
209                         visited[v] = 1;
210                         Q.push (v);
211                         cost[v] = cost[u] + 1;
212                     }
213                 }
214             }
215         }
216         /////////////////////////// BFS END ///////////////////////////
217
218
219         /////////////////////////// DIJKSTRA ///////////////////////////
220
221         //See                                                                    ↵
            /media/jenin/Softwares/Programming/UVa/Accepted/Sending_email-10986.cpp
222         //
223         //typedef pair<int, int> pii;
```

```
225         //typedef vector <int> vi;
226         //graph declaration
227         //vector<vpii> graph (n, vpii());
228
229         void dijkstra (int s, vi *dist, vector<vpii> *graph) {
230             priority_queue< pii, vector<pii>, greater<pii> > pq;
231             pq.push (pii (0, s) );
232
233             while (!pq.empty() ) {
234                 pii front = pq.top();
235                 pq.pop();
236                 int d = front.first, u = front.second;
237
238                 if (d == dist->at (u) ) {
239                     for (int j = 0; j < (int) graph->at (u).size(); j++) {
240                         pii v = graph->at (u) [j];                    // all
                            outgoing edges from u
241
242                         if (dist->at (u) + v.second < dist->at (v.first) ) {
243                             dist->at (v.first) = dist->at (u) + v.second; //
                                relax operation
244                             pq.push (pii (dist->at (v.first), v.first) );
245                         }
246                     }
247                 }
248             }
249         }
250
251         ////////////////////////// DIJKSTRA END //////////////////////////
252
253 };
254
255 ////////////////////////// UNION FIND //////////////////////////
256 class UNION_FIND {
257         int parent[26];
258
259         void Make_Set (int x) {
260             parent[x] = x;
261         }
262
263         int Find (int x) {
264             if (x != parent[x]) parent[x] = Find (parent[x]);
265
266             return parent[x];
267         }
268
269         void Union (int &x, int &y) {
270             int PX = Find (x), PY = Find (y);
271             parent[PX] = PY;
272         }
273         // example
274         // if (Find(x) != Find(y)){
275         //     Union(x, y);
276         // }
277 };
278 ////////////////////////// UNION FIND END //////////////////////////
279
```

```cpp

////////////////////////// DFS //////////////////////////

class DFS {
        vector<int> adj [100]; // for BFS & bfs_vis
        int finish[100], discover[100], color[100], time_cnt = 0;
        //discovery time and finishing time for DFS

        void dfs (int u) {
            int i, v, sz;
            time_cnt++;
            discover[u] = time_cnt;
            color[u] = 1;   ///Gray = visiting = 1, black = visited = 2;
            sz = (int) adj[u].size();

            for (i = 0; i < sz; i++) {
                v = adj[u][i];

                if (color[v] == 0) {
                    dfs (v);
                }
            }

            color[u] = 2;
            time_cnt++;
            finish[u] = time_cnt;
        }
};
////////////////////////// DFS END //////////////////////////

class Topsort {
        int indegree[MAX];
        vector<int> adj[MAX], tops;

        // if 0 returned, topsort is not possible, 1 otherwise
        // order has been kept in tops
        // adj[] inserted in this way, u to v then asj[u].pb(v)
        // top sort relation is not bidirectional
        // cycle checking not needed
        // Top sort is not possible in DAG - Directed Acyclic Graph, 0 returned
        bool topsort (int size) {
            int k, i, v;

            for (k = 1; k <= size; k++) {
                for (i = 1; i <= size; i++) {
                    if (indegree[i] == 0) {
                        tops.push_back (i);
                        v = (int) adj[i].size();

                        for (int j = 0; j < v; j++) {
                            indegree[adj[i][j]]--;
                        }

                        indegree[i]--;
                        break;
                    }
                }
```

```
337
338                 if (i > size) return 0;
339             }
340
341             return 1;
342         }
343 };
344
345 ////////////////////////// SIEVE ////////////////////////
346 class Primes {
347         bool isPrime[MAXS]; //for sieve
348         int prime[MAXS]; //for sieve
349
350     public:
351         bool isprime (int num) {
352             if (num == 2) return true;
353
354             if (num < 2 or num % 2 == 0) return false;
355
356             int i, root = (int) sqrt (num);
357
358             for (i = 3; i <= root; i += 2)
359                 if (num % i == 0)    return false;
360
361             return true;
362         }
363
364         int sieve (int n) {
365             int i, res, j;
366             double root = sqrt (n);
367             isPrime[0] = isPrime[1] = 1;
368
369             for (i = 4; i < n; i += 2)
370                 isPrime[i] = 1;
371
372             for (i = 3, j = 0; i <= root; i += 2) {
373                 if (!isPrime[i]) {
374                     for (j = i * i; j < n; j += 2 * i)
375                         isPrime[j] = 1;
376                 }
377             }
378
379             for (i = 0, res = 0; i < n; i++) {
380                 if (isPrime[i] == 0) {
381                     prime[res++] = i;
382                 }
383             }
384
385             return (res - 1);
386         }
387
388
389         // bitwise seive
390         //int isPrime[MAXS/32]; //for sieve
391         //int prime[MAXS]; //for sieve
392
393         int bit_sieve (int n) {
```

```
394             int i, res, j;
395             double root = sqrt (n);
396             bit_on (isPrime[0], 0);
397             bit_on (isPrime[0], 1);
398 
399             for (i = 4; i < n; i += 2)
400                 bit_on (isPrime[i >> 5], i & 31);
401 
402             for (i = 3, j = 0; i <= root; i += 2) {
403                 if (bitchk (isPrime[i >> 5], i & 31) == 0) {
404                     for (j = i * i; j < n; j += 2 * i)
405                         bit_on (isPrime[j >> 5], j & 31);
406                 }
407             }
408 
409             for (i = 0, res = 0; i < n; i++) {
410                 if (bitchk (isPrime[i >> 5], i & 31) == 0) {
411                     prime[res++] = i;
412                 }
413             }
414 
415             return (res - 1);
416         }
417 
418         // Segmented Sieve of Eratosthenes
419 
420         int segsieve[100000];
421 
422         ///Returns number of primes between segment [a,b]
423         int segmentedSieve ( int a, int b, int primesize) {
424             if ( a == 1 ) a++;
425 
426             int sqrtn = (int) sqrt (b);
427             memset ( segsieve, 0, sizeof segsieve ); ///Make all index of
                segsieve 0.
428 
429             // maxprime is used in bit sieve or normal seive code
430             for ( int i = 0; i < primesize && prime[i] <= sqrtn; i++ ) {
431                 int p = prime[i];
432                 int j = p * p;
433 
434                 ///If j is smaller than a, then shift it inside of segment [a,b]
435                 if ( j < a ) j = ( ( a + p - 1 ) / p ) * p;
436 
437                 for ( ; j <= b; j += p ) {
438                     segsieve[j - a] = 1; ///mark them as not prime
439                 }
440             }
441 
442             int res = 0;
443 
444             for ( int i = a; i <= b; i++ ) {
445                 ///If it is not marked, then it is a prime
446                 if ( segsieve[i - a] == 0 ) res++;
447             }
448 
449             return res;
```

```
451  };
452  /////////////////////////// SIEVE END ///////////////////////////
453
454
455  /////////////////////////// PRIME FACTOR ///////////////////////////
456  class Prime_fact {
457          int fact[100][2]; //for prime factor
458          int isPrime[MAX], prime[MAX];  // Primes class
459          //100 will be replaced with max number of factors
460
461      public:
462          int primefactor (long long n, int primesize) {
463              // res is return value of seive
464              int i, j;
465
466              if (n == 0 || isPrime[n] == 0)
467                  return 0;
468
469              for (i = 0, j = 0; i < primesize; i++) {
470                  if (n % prime[i] == 0) {
471                      fact[j][0] = prime[i];
472                      fact[j][1] = 0;
473
474                      while (n % prime[i] == 0) {
475                          n /= prime[i];
476                          fact[j][1]++;
477                      }
478
479                      j++;
480                  }
481              }
482
483              return j;
484          }
485  };
486
487  /////////////////////////// EULER'S TOTIENT FUNCTION ///////////////////////
488
489  class Totient {
490
491      public:
492          // phi[i] stores euler totient function for i
493          // gcdsum[j] stores result for value j
494          int phi[MAX];
495          int gcdsum[MAX];
496
497          /// Computes and prints totien of all numbers
498          /// smaller than or equal to n.
499          /// http://www.geeksforgeeks.org/
500          /// eulers-totient-function-for-all-numbers-smaller-than-or-equal-to-n/
501          void compute (int n) {
502              // Create and initialize an array to store
503              // phi or totient values
504              //long long phi[n + 1];
505              for (int i = 1; i <= n; i++)
506                  phi[i] = i; // indicates not evaluated yet
507
```

```
508                 // and initializes for product
509                 // formula.
510
511                 // Compute other Phi values
512                 for (int p = 2; p <= n; p++) {
513                     //If phi[p] is not computed already,
514                     //then number p is prime
515                     if (phi[p] == p) {
516                         // Phi of a prime number p is
517                         // always equal to p-1.
518                         phi[p] = p - 1;
519
520                         // update phi values of all
521                         // multiples of p
522                         for (int i = 2 * p; i <= n; i += p) {
523                             // Add contribution of p to its
524                             // multiple i by multiplying with
525                             // (1 - 1/p)
526                             phi[i] = (phi[i] / p) * (p - 1);
527                         }
528                     }
529                 }
530             }
531
532             //Precomputes result for all numbers till MAX
533             void sumOfGcdPairs (int n) {
534                 // Precompute all phi value
535                 compute (n);
536
537                 //gcdsum[0] = 0;
538                 for (int i = 1; i < n; ++i) {
539                     // Iterate throght all the divisors
540                     // of i.
541                     for (int j = 2; i * j < n; ++j)
542                         gcdsum[i * j] += i * phi[j];
543                 }
544
545                 // Add summation of previous calculated sum
546                 for (int i = 2; i < n; i++)
547                     gcdsum[i] += gcdsum[i - 1];
548             }
549
550             // http://www.geeksforgeeks.org/eulers-totient-function/
551             int phi_single (int n) {
552                 int res = n;   // Initialize result as n
553
554                 // Consider all prime factors of n and subtract their
555                 // multiples from result
556                 for (int p = 2; p * p <= n; ++p) {
557                     // Check if p is a prime factor.
558                     if (n % p == 0) {
559                         // If yes, then update n and result
560                         while (n % p == 0)
561                             n /= p;
562
563                         res -= res / p;
564                     }
```

```
565             }
566
567             // If n has a prime factor greater than sqrt(n)
568             // (There can be at-most one such prime factor)
569             if (n > 1)
570                 res -= res / n;
571
572             return res;
573         }
574 };
575
576 ////////////////////////// EULER'S TOTIENT FUNCTION ////////////////////////
577
578
579 ////////////////////////////// MOBIOUS FUNCTION ////////////////////////////
580
581 class Mobious {
582
583         int mu[MAX];
584
585         void func (int n) {
586             int fact[100][2]; //for prime factor, declared in Prime_fact class
587             Primes primes;
588             Prime_fact prime_fact;
589             int num_of_unique_factors, fl;
590             int primesz = primes.sieve (MAXS);
591
592             for (int i = 1; i <= n; i++) {
593                 num_of_unique_factors = prime_fact.primefactor (i, primesz);
594                 fl = 2;
595
596                 for (int j = 0; j < num_of_unique_factors; j++) {
597                     if (fact[j][1] > 1) {
598                         fl = 0;
599                         break;
600                     }
601                 }
602
603                 if (fl == 0) mu[i] = 0;
604                 else {
605                     if (num_of_unique_factors & 1)
606                         mu[i] = -1;
607                     else mu[i] = 1;
608                 }
609             }
610         }
611 };
612
613 ////////////////////////////// MOBIOUS FUNCTION ////////////////////////////
614
615 void swap (int *x, int *y) {
616     // Code to swap 'x' and 'y'
617     *x = *x * *y; // x now becomes 15
618     *y = *x / *y; // y becomes 10
619     *x = *x / *y; // x becomes 5
620 }
621
```

```
622
623 long double fibbonacci (long double f) {
624     return (pow ( ( (1 + sqrt (5) ) / 2), f) - pow ( ( (1 - sqrt (5) ) / 2),
625             f) ) / sqrt (5);
626 }
627
628
629 int gcd (int a, int b)  {
630     if (b == 0) return a;
631     else return gcd (b, a % b);
632 }
633
634
635 // http://www.geeksforgeeks.org/euclidean-algorithms-basic-and-extended/
636 int gcdExtended (int a, int b, int *x, int *y) {
637     // Base Case
638     if (a == 0) {
639         *x = 0;
640         *y = 1;
641         return b;
642     }
643
644     int x1, y1; // To store results of recursive call
645     int gcd = gcdExtended (b % a, a, &x1, &y1);
646     // Update x and y using results of recursive
647     // call
648     *x = y1 - (b / a) * x1;
649     *y = x1;
650     return gcd;
651 }
652
653 int lcm (int a, int b) {
654     return (a * b) / gcd (a, b);
655 }
656
657
658 int lcm2 (int a, int b) {
659     int temp = a;
660
661     while (1) {
662         if (temp % b == 0 && temp % a == 0) break;
663
664         temp++;
665     }
666
667     return temp;
668 }
669
670 char *strrev (char *str) { //Used by big_int_sum & big_int_mul
671     char *p1, *p2;
672
673     if (! str || ! *str) return str;
674
675     for (p1 = str, p2 = str + strlen (str) - 1; p2 > p1; ++p1, --p2) {
676         *p1 ^= *p2;
677         *p2 ^= *p1;
678         *p1 ^= *p2;
```

```
679         }
680 
681         return str;
682     }
683 
684     ////////////////////////// BIG NUMBER //////////////////////////
685     class Big_num {
686             int sum_s (int a, int b, int c) { //Used by big_int_sum & big_int_mul
687                 if (a + b + c > 9) return (a + b + c) % 10;
688 
689                 return a + b + c;
690             }
691             int sum_c (int a, int b, int c) { //Used by big_int_sum & big_int_mul
692                 if (a + b + c > 9) return (a + b + c) / 10;
693 
694                 return 0;
695             }
696             int mul_s (int a, int b, int c) { //Used by big_int_mul
697                 if (a * b + c > 9) return (a * b + c) % 10;
698 
699                 return a * b + c;
700             }
701             int mul_c (int a, int b, int c) { //Used by big_int_mul
702                 if (a * b + c > 9) return (a * b + c) / 10;
703 
704                 return 0;
705             }
706 
707             void big_int_sum (char *a, char *b, char *c) {  // a + b = c
708                 int len_a, len_b, i, j, k, carry;
709                 len_a = (int) strlen (a);
710                 len_b = (int) strlen (b);
711                 carry = 0;
712                 k = 0, i = len_a - 1, j = len_b - 1;
713 
714                 for (; i >= 0 && j >= 0; j--, i--, k++) {
715                     c[k] = (char) (sum_s (a[i] - 48, b[j] - 48, carry) + 48);
716                     carry = sum_c (a[i] - 48, b[j] - 48, carry);
717                 }
718 
719                 if (i >= 0 && i != j)
720                     for (; i >= 0; i--, k++) {
721                         c[k] = (char) (sum_s (a[i] - 48, 0, carry) + 48);
722                         carry = sum_c (a[i] - 48, 0, carry);
723                     } else if (j >= 0 && i != j)
724                     for (; j >= 0; j--, k++) {
725                         c[k] = (char) (sum_s (0, b[j] - 48, carry) + 48);
726                         carry = sum_c (0, b[j] - 48, carry);
727                     }
728 
729                 if (carry != 0) c[k++] = (char) (carry + 48);
730 
731                 c[k] = '\0';
732                 strrev (c);
733             }
734 
735 
```

```
736
737         void big_int_mul (char *x, char *y, char *t) {
738             int i, j, k, C, r, xi, yi, Cs, m;
739             C = 0, Cs = 0;
740
741             for (i = 0;; i++) if (x[i] == '\0') break;
742
743             xi = i - 1;
744
745             for (i = 0;; i++) if (y[i] == '\0') break;
746
747             yi = i - 1;
748
749             for (i = 0; i < 502; i++) t[i] = '0';
750
751             for (j = 0; yi >= 0; yi--, j++) {
752                 for (i = 0, k = xi; k >= 0; k--, i++) {
753                     r = mul_s (x[k] - 48, y[yi] - 48, C);
754                     C = mul_c (x[k] - 48, y[yi] - 48, C);
755                     m = t[i + j];
756                     t[i + j] = (char) (sum_s (r, m - 48, Cs) + 48);
757                     Cs = sum_c (r, m - 48, Cs);
758                 }
759
760                 t[i + j] = (char) (C + Cs + 48);
761                 C = 0;
762                 Cs = 0;
763             }
764
765             for (k = i + j; k > 0; k--) {
766                 if (t[k] != 48) {
767                     t[k + 1] = 0;
768                     break;
769                 }
770             }
771
772             t[k + 1] = 0;
773             strrev (t);
774         }
775
776
777
778         unsigned long long big_int_div (char *a, unsigned long long b, char *c) {
779             int la;
780             int i, k = 0, flag = 0;
781             unsigned long long temp = 1, reminder;
782             la = (int) strlen (a);
783
784             for (i = 0; i <= la; i++) a[i] = (char) (a[i] - 48);
785
786             temp = a[0];
787             reminder = a[0];
788
789             for (i = 1; i <= la; i++) {
790                 if (b <= temp) {
791                     c[k++] = (char) (temp / b);
792                     temp = temp % b;
```

```
793                         reminder = temp;
794                         temp = temp * 10 + a[i];
795                         flag = 1;
796                     } else {
797                         reminder = temp;
798                         temp = temp * 10 + a[i];
799 
800                         if (flag == 1) c[k++] = 0;
801                     }
802                 }
803 
804                 for (i = 0; i < k; i++) {
805                     c[i] = (char) (c[i] + '0');
806                 }
807 
808                 c[i] = '\0';
809 
810                 if (i == 0) {
811                     c[i] = '0';
812                     c[1] = '\0';
813                 }
814 
815                 return reminder;
816             }
817 
818             // Multiplies str1 and str2, and prints result.
819             // Time Complexity : O(m*n), where m and n are length of two number that
820             // need to be multiplied.
821             // ONLY NON-NEGATIVE
822             void bigInt_mult (string &num1, string &num2, string &ans) {
823                 int n1 = (int) num1.size();
824                 int n2 = (int) num2.size();
825 
826                 if (n1 == 0 || n2 == 0) {
827                     ans = "0";
828                     return;
829                 }
830 
831                 // will keep the result number in vector
832                 // in reverse order
833                 vector<int> result (n1 + n2, 0);
834                 // Below two indexes are used to find positions
835                 // in result.
836                 int i_n1 = 0;
837                 int i_n2 = 0;
838 
839                 // Go from right to left in num1
840                 for (int i = n1 - 1; i >= 0; i--) {
841                     int carry = 0;
842                     int n1_next = num1[i] - '0';
843                     // To shift position to left after every
844                     // multiplication of a digit in num2
845                     i_n2 = 0;
846 
847                     // Go from right to left in num2
848                     for (int j = n2 - 1; j >= 0; j--) {
849                         // Take current digit of second number
```

```
850                     int n2_next = num2[j] - '0';
851                     // Multiply with current digit of first number
852                     // and add result to previously stored result
853                     // at current position.
854                     int sum = n1_next * n2_next + result[i_n1 + i_n2] + carry;
855                     // Carry for next iteration
856                     carry = sum / 10;
857                     // Store result
858                     result[i_n1 + i_n2] = sum % 10;
859                     i_n2++;
860                 }
861 
862                 // store carry in next cell
863                 if (carry > 0)
864                     result[i_n1 + i_n2] += carry;
865 
866                 // To shift position to left after every
867                 // multiplication of a digit in num1.
868                 i_n1++;
869             }
870 
871             // ignore '0's from the right
872             int i = (int) result.size() - 1;
873 
874             while (i >= 0 && result[i] == 0)
875                 i--;
876 
877             // If all were '0's - means either both or
878             // one of num1 or num2 were '0'
879             if (i == -1) {
880                 ans = "0";
881                 return;
882             }
883 
884             // generate the result string
885             ans.clear();
886 
887             while (i >= 0)
888                 ans += std::to_string (result[i--]);
889         }
890 
891         // Function for finding sum of larger numbers
892         // https://www.geeksforgeeks.org/sum-two-large-numbers/
893         // Time Complexity : O(n1 + n2) where n1 and n2 are lengths of two input
894         // strings representing numbers.
895         // ONLY NON-NEGATIVE
896         void bigInt_sum (string &str1, string &str2, string &ans) {
897             // Before proceeding further, make sure length
898             // of str2 is larger.
899             if (str1.length() > str2.length() )
900                 swap (str1, str2);
901 
902             // Take an empty string for storing result
903             ans = "";
904             // Calculate lenght of both string
905             int n1 = (int) str1.length(), n2 = (int) str2.length();
906             int diff = n2 - n1;
```

```
907             // Initialy take carry zero
908             int carry = 0;
909 
910             // Traverse from end of both strings
911             for (int i = n1 - 1; i >= 0; i--) {
912                 // Do school mathematics, compute sum of
913                 // current digits and carry
914                 int sum = ( (str1[i] - '0') +
915                             (str2[i + diff] - '0') +
916                             carry);
917                 ans.push_back ( (char) (sum % 10 + '0') );
918                 carry = sum / 10;
919             }
920 
921             // Add remaining digits of str2[]
922             for (int i = n2 - n1 - 1; i >= 0; i--) {
923                 int sum = ( (str2[i] - '0') + carry);
924                 ans.push_back ( (char) (sum % 10 + '0') );
925                 carry = sum / 10;
926             }
927 
928             // Add remaining carry
929             if (carry)
930                 ans.push_back ( (char) (carry + '0') );
931 
932             // reverse resultant string
933             reverse (ans.begin(), ans.end() );
934         }
935 };
936 ////////////////////////// BIG NUMBER END //////////////////////////
937 
938 bool isleapyear (long int year) {
939     /*Is year divided by 4? After every 100 years a leap year is not
940     counted and after every 400 years we count leap year*/
941     if ( ( ( (year % 4) == 0) && ( (year % 100) != 0) ) || ( (year % 400) == 0) )
942         return true;
943     else return false;
944 }
945 
946 
947 int big_mod (int base, int power, int mod) {
948     if (power == 0)    return 1;
949     else if (power % 2 == 1) {
950         int p1 = base % mod;
951         int p2 = (big_mod (base, power - 1, mod) ) % mod;
952         return (p1 * p2) % mod;
953     } else if (power % 2 == 0) {
954         int p1 = (big_mod (base, power / 2, mod) ) % mod;
955         return (p1 * p1) % mod;
956     }
957 
958     return 0;
959 }
960 
961 
962 int binarysearch (int *array, int end, int key) {
963     //array must be sorted, if key is found return 0 based index,
```

```
964       //else -1;
965       int start = 0, mid;
966       end--;
967
968       while (start <= end) {
969           mid = (start + end) / 2;
970
971           if (key == array[mid]) return mid;
972           else if (key < array[mid]) end = mid - 1;
973           else start = mid + 1;
974       }
975
976       return -1;
977   }
978
979   ////////////////////////////// SORTING /////////////////////////////
980
981   class Sorting {
982           void bubble_sort (int *list, int n) {
983               int c, d, t;
984
985               for (c = 0 ; c < ( n - 1 ); c++) {
986                   for (d = 0 ; d < n - c - 1; d++) {
987                       if (list[d] > list[d + 1]) {
988                           /* Swapping */
989                           t         = list[d];
990                           list[d]   = list[d + 1];
991                           list[d + 1] = t;
992                       }
993                   }
994               }
995           }
996
997
998           void insertion_sort (int *array, int n) {
999               int t, d, c;
1000
1001              for (c = 1 ; c <= n - 1; c++) {
1002                  d = c;
1003
1004                  while ( d > 0 && array[d] < array[d - 1]) {
1005                      t          = array[d];
1006                      array[d]   = array[d - 1];
1007                      array[d - 1] = t;
1008                      d--;
1009                  }
1010              }
1011          }
1012
1013
1014          // Merge sort
1015          void merge (int *a, int low, int high, int mid) { //used by merge_sort
1016              int i, j, k, c[50];
1017              i = low;
1018              k = low;
1019              j = mid + 1;
1020
```

```
1021                while (i <= mid && j <= high) {
1022                    if (a[i] < a[j]) {
1023                        c[k] = a[i];
1024                        k++;
1025                        i++;
1026                    } else {
1027                        c[k] = a[j];
1028                        k++;
1029                        j++;
1030                    }
1031                }
1032
1033                while (i <= mid) {
1034                    c[k] = a[i];
1035                    k++;
1036                    i++;
1037                }
1038
1039                while (j <= high) {
1040                    c[k] = a[j];
1041                    k++;
1042                    j++;
1043                }
1044
1045                for (i = low; i < k; i++) a[i] = c[i];
1046            }
1047
1048            void merge_sort (int *a, int low, int high) { //low and high inclusive
1049                int mid;
1050
1051                if (low < high) {
1052                    mid = (low + high) / 2;
1053                    merge_sort (a, low, mid);
1054                    merge_sort (a, mid + 1, high);
1055                    merge (a, low, high, mid);
1056                }
1057
1058                return;
1059            }
1060
1061
1062            void quick_sort (int *array, int start, int end) {
1063                //start and end inclusive
1064                int low = start, high = start, i;
1065
1066                if (start < end) {
1067                    for (i = start; i < end; i++) {
1068                        if (array[i] < array[end]) {
1069                            swap (array[i], array[low]);
1070                            low++;
1071                            high++;
1072                        } else high++;
1073                    }
1074
1075                    swap (array[end], array[low]);
1076                    quick_sort (array, start, low - 1);
1077                    quick_sort (array, low + 1, high);
```

```
1078                } else return;
1079            }
1080
1081
1082            // Heap sort start
1083            // How to use
1084            // First call Build_MaxHeap(n), where n in size of array arr
1085            //      Build_MaxHeap(n);
1086            // then call HeapSort(n);
1087            //      HeapSort(n);
1088
1089            int arr[100];
1090            void MakeHeap (int i, int n) {
1091                int j, temp;
1092                temp = arr[i];
1093                j = 2 * i;
1094
1095                while (j <= n) {
1096                    if (j < n && arr[j + 1] > arr[j]) {
1097                        j++;
1098                    }
1099
1100                    if (temp > arr[j])
1101                        break;
1102                    else if (temp <= arr[j]) {
1103                        arr[j / 2] = arr[j];
1104                        j = 2 * j;
1105                    }
1106                }
1107
1108                arr[j / 2] = temp;
1109            }
1110
1111            void HeapSort (int n) {
1112                int i, temp;
1113
1114                for (i = n; i >= 2; i--) {
1115                    temp = arr[i];
1116                    arr[i] = arr[1];
1117                    arr[1] = temp;
1118                    MakeHeap (1, i - 1);
1119                }
1120            }
1121
1122            void Build_MaxHeap (int n) {
1123                int i;
1124
1125                for (i = n / 2; i >= 1; i--) {
1126                    MakeHeap (i, n);
1127                }
1128            }
1129            // Heap sort end
1130    };
1131    ////////////////////////////// SORTING //////////////////////////////
1132
1133    ////////////// MOST SIGNIFICANT DIGIT //////////////
1134
```

```
1135  //Return most significant digit
1136  uint32_t powers_of_10[33] = {
1137      1000000000, 1000000000,
1138      100000000, 100000000, 100000000,
1139      10000000, 10000000, 10000000,
1140      1000000, 1000000, 1000000, 1000000,
1141      100000, 100000, 100000,
1142      10000, 10000, 10000,
1143      1000, 1000, 1000, 1000,
1144      100, 100, 100,
1145      10, 10, 10,
1146      1, 1, 1, 1, 1
1147  };
1148
1149  int CalcFirstDecimalDigit (uint32_t x) {
1150      int leading_zeros = __builtin_clz (x);
1151      x /= powers_of_10[leading_zeros];
1152
1153      if (x >= 10)
1154          return 1;
1155      else
1156          return x;
1157  }
1158
1159
1160  int most_significant_digit (int n) {
1161      double K = log10 (n);
1162      K = K - floor (K);
1163      int X = (int) pow (10, K);
1164      return X;
1165  }
1166
1167  ////////////// MOST SIGNIFICANT DIGIT END ////////////
1168
1169  //https://www.geeksforgeeks.org/count-sum-of-digits-in-numbers-from-1-to-n/
1170  // Function to computer sum of digits in numbers from 1 to n
1171  // Comments use example of 328 to explain the code
1172  int sumOfDigitsFrom1ToN (int n) {
1173      // base case: if n<10 return sum of
1174      // first n natural numbers
1175      if (n < 10)
1176          return n * (n + 1) / 2;
1177
1178      // d = number of digits minus one in n. For 328, d is 2
1179      int d = log10 (n);
1180      // computing sum of digits from 1 to 10^d-1,
1181      // d=1 a[0]=0;
1182      // d=2 a[1]=sum of digit from 1 to 9 = 45
1183      // d=3 a[2]=sum of digit from 1 to 99 = a[1]*10 + 45*10^1 = 900
1184      // d=4 a[3]=sum of digit from 1 to 999 = a[2]*10 + 45*10^2 = 13500
1185      int *a = new int[d + 1];
1186      a[0] = 0, a[1] = 45;
1187
1188      for (int i = 2; i <= d; i++)
1189          a[i] = a[i - 1] * 10 + 45 * ceil (pow (10, i - 1) );
1190
1191      // computing 10^d
```

```
1192        int p = ceil (pow (10, d) );
1193        // Most significant digit (msd) of n,
1194        // For 328, msd is 3 which can be obtained using 328/100
1195        int msd = n / p;
1196        // EXPLANATION FOR FIRST and SECOND TERMS IN BELOW LINE OF CODE
1197        // First two terms compute sum of digits from 1 to 299
1198        // (sum of digits in range 1-99 stored in a[d]) +
1199        // (sum of digits in range 100-199, can be calculated as 1*100 + a[d]
1200        // (sum of digits in range 200-299, can be calculated as 2*100 + a[d]
1201        //  The above sum can be written as 3*a[d] + (1+2)*100
1202        // EXPLANATION FOR THIRD AND FOURTH TERMS IN BELOW LINE OF CODE
1203        // The last two terms compute sum of digits in number from 300 to 328
1204        // The third term adds 3*29 to sum as digit 3 occurs in all numbers
1205        //                from 300 to 328
1206        // The fourth term recursively calls for 28
1207        return msd * a[d] + (msd * (msd - 1) / 2) * p +
1208               msd * (1 + n % p) + sumOfDigitsFrom1ToN (n % p);
1209    }
1210
1211    //www.geeksforgeeks.org/finding-sum-of-digits-of-a-number-until-sum-becomes-single-digit
1212    //Finding sum of digits of a number until sum becomes single digit
1213    int digSum (int n) {
1214        if (n == 0)
1215            return 0;
1216
1217        return (n % 9 == 0) ? 9 : (n % 9);
1218    }
1219
1220    template <class T>
1221    int numDigits (T number) {
1222        int digits = 0;
1223
1224        if (number < 0) digits = 1;
1225
1226        while (number) {
1227            number /= 10;
1228            digits++;
1229        }
1230
1231        return digits;
1232    }
1233
1234
1235
1236
1237    ////////////////////////// SPARSE TABLE ////////////////////////
1238
1239    class Sparse_table {
1240            //#define Max 10000005
1241            int ST[24][MAX];
1242            int Array[MAX];
1243        public:
1244            Sparse_table (int N) {
1245                for (int i = 0; i < N; i++) ST[0][i] = i;
1246
1247                for (int k = 1; (1 << k) < N; k++) {
```

```
1248                    for (int i = 0; i + (1 << k) <= N; i++) {
1249                        int x = ST[k - 1][i];
1250                        int y = ST[k - 1][i + (1 << (k - 1) )];
1251                        ST[k][i] = Array[x] <= Array[y] ? x : y;
1252                    }
1253                }
1254            }
1255
1256            int RMQ (int i, int j) {
1257                // return min value index number of Array from i to j, including
1258                //if(i == j) return i;
1259                int k = (int) log2 (j - i);
1260                int x = ST[k][i];
1261                int y = ST[k][j - (1 << k) + 1];
1262                return Array[x] <= Array[y] ? x : y;
1263            }
1264    };
1265    ////////////////////////// SPARSE TABLE END //////////////////////////
1266
1267
1268    ////////////////////////// INIX TO POSTFIX //////////////////////////
1269
1270    //deque: Every element of the equation is separate string in the deque
1271
1272    vector<string> infix_to_postfix (deque<string> v) {
1273        vector<string> P;
1274        stack<string> Stk;
1275        v.push_front ("(");
1276        v.push_back (")");
1277
1278        for (int i = 0 ; i < (int) v.size() ; i++) {
1279            string tmp = v[i];
1280
1281            /// Case 1 : number
1282            if (tmp[0] >= '0' && tmp[0] <= '9') {
1283                P.push_back (tmp);
1284            }
1285            /// Operator
1286            else if (tmp == "+" || tmp == "-" || tmp == "*" || tmp == "/") {
1287                if (tmp == "+" || tmp == "-") {
1288                    while (!Stk.empty() && (Stk.top() == "+" || Stk.top() == "-"
1289                                            || Stk.top() == "*" || Stk.top() == "/"
                                                ) ) {
1290                        P.push_back (Stk.top() );
1291                        Stk.pop();
1292                    }
1293                } else if (tmp == "*" || tmp == "/" ) {
1294                    while (!Stk.empty() && \
1295                           ( Stk.top() == "*" || Stk.top() == "/") ) {
1296                        P.push_back (Stk.top() );
1297                        Stk.pop();
1298                    }
1299                }
1300
1301                Stk.push (tmp);
1302            } else {
1303                if (tmp == "(") Stk.push (tmp);
```

```
1305                    while (Stk.top() != "(") {
1306                        P.push_back (Stk.top() );
1307                        Stk.pop();
1308                    }
1309
1310                    if (!Stk.empty() ) Stk.pop();
1311                }
1312            }
1313        }
1314
1315        return P;
1316    }
1317
1318    ////////////////////////// INIX TO POSTFIX //////////////////////////
1319
1320
1321
1322    ////////////////////////// IS SUBSECUENCE //////////////////////////
1323
1324    class Subsecuence {
1325            // Returns true if str1[] is a subsequence of str2[]. m is
1326            // length of str1 and n is length of str2
1327            bool isSubSequence (char str1[], char str2[], int m, int n) {
1328                // Base Cases
1329                if (m == 0) return true;
1330
1331                if (n == 0) return false;
1332
1333                // If last characters of two strings are matching
1334                if (str1[m - 1] == str2[n - 1])
1335                    return isSubSequence (str1, str2, m - 1, n - 1);
1336
1337                // If last characters are not matching
1338                return isSubSequence (str1, str2, m, n - 1);
1339            }
1340
1341
1342            // Returns true if str1[] is a subsequence of str2[]. m is
1343            // length of str1 and n is length of str2
1344            bool isSubSequence_it (char str1[], char str2[], int m, int n) {
1345                int j = 0; // For index of str1 (or subsequence
1346
1347                // Traverse str2 and str1, and compare current character
1348                // of str2 with first unmatched char of str1, if matched
1349                // then move ahead in str1
1350                for (int i = 0; i < n && j < m; i++)
1351                    if (str1[j] == str2[i])
1352                        j++;
1353
1354                // If all characters of str1 were found in str2
1355                return (j == m);
1356            }
1357
1358            ////////////////////////// IS SUBSECUENCE //////////////////////////
1359
1360
1361
```

```
1362            ////////////////////////// LCS //////////////////////////
1363
1364            int dp[1010][1010];
1365            /* Returns length of LCS for X[0..m-1], Y[0..n-1] */
1366            // last to first, m and n is size of strings
1367            int lcs ( char *X, char *Y, int m, int n ) {
1368                if (m == 0 || n == 0)
1369                    return 0;
1370
1371                if (dp[m][n] != -1)
1372                    return dp[m][n];
1373
1374                if (X[m - 1] == Y[n - 1])
1375                    return dp[m][n] = 1 + lcs (X, Y, m - 1, n - 1);
1376                else
1377                    return dp[m][n] = max (lcs (X, Y, m, n - 1), lcs (X, Y, m - 1,
                        n) );
1378            }
1379
1380
1381            // dp array needed, last to first
1382            string ans; // store lcs here
1383            void lcs_print (string X, string Y, int m, int n) {
1384                if (m == -1 or n == -1) {
1385                    reverse (all (ans) );
1386                    cout << ans << endl;
1387                    return;
1388                }
1389
1390                if (X[m - 1] == Y[n - 1]) {
1391                    ans += X[m - 1];
1392                    lcs_print (X, Y, m - 1, n - 1);
1393                } else {
1394                    if (dp[m - 1][n] > dp[m][n - 1]) lcs_print (X, Y, m - 1, n);
1395                    else lcs_print (X, Y, m, n - 1);
1396                }
1397            }
1398
1399
1400            /* Returns length of LCS for X[0..m-1], Y[0..n-1] */
1401            int lcs_it ( char *X, char *Y, int m, int n ) {
1402                int i, j;
1403
1404                // Following steps build L[m+1][n+1] in bottom up fashion. Note
1405                // that L[i][j] contains length of LCS of X[0..i-1] and Y[0..j-1]
1406                for (i = 0; i <= m; i++) {
1407                    for (j = 0; j <= n; j++) {
1408                        if (i == 0 || j == 0)
1409                            dp[i][j] = 0;
1410                        else if (X[i - 1] == Y[j - 1])
1411                            dp[i][j] = dp[i - 1][j - 1] + 1;
1412                        else
1413                            dp[i][j] = max (dp[i - 1][j], dp[i][j - 1]);
1414                    }
1415                }
1416
1417                /* dp[m][n] contains length of LCS for X[0..n-1] and Y[0..m-1] */
```

```
1419            }
1420
1421
1422            // LCS PRINT
1423
1424            void lcs_print ( char *X, char *Y, int m, int n ) {
1425                //int dp[m + 1][n + 1]; // needed, declare before
1426
1427                // Following steps build L[m+1][n+1] in bottom up fashion. Note
1428                // that L[i][j] contains length of LCS of X[0..i-1] and Y[0..j-1]
1429                for (int i = 0; i <= m; i++) {
1430                    for (int j = 0; j <= n; j++) {
1431                        if (i == 0 || j == 0)
1432                            dp[i][j] = 0;
1433                        else if (X[i - 1] == Y[j - 1])
1434                            dp[i][j] = dp[i - 1][j - 1] + 1;
1435                        else
1436                            dp[i][j] = max (dp[i - 1][j], dp[i][j - 1]);
1437                    }
1438                }
1439
1440                // Following code is used to print LCS
1441                int index = dp[m][n];
1442                // Create a character array to store the lcs string
1443                char lcs[index + 1];
1444                lcs[index] = '\0'; // Set the terminating character
1445                // Start from the right-most-bottom-most corner and
1446                // one by one store characters in lcs[]
1447                int i = m, j = n;
1448
1449                while (i > 0 && j > 0) {
1450                    // If current character in X[] and Y are same, then
1451                    // current character is part of LCS
1452                    if (X[i - 1] == Y[j - 1]) {
1453                        // Put current character in result
1454                        lcs[index - 1] = X[i - 1];
1455                        // reduce values of i, j and index
1456                        i--; j--; index--;
1457                    }
1458                    // If not same, then find the larger of two and
1459                    // go in the direction of larger value
1460                    else if (dp[i - 1][j] > dp[i][j - 1])
1461                        i--;
1462                    else
1463                        j--;
1464                }
1465
1466                // Print the lcs
1467                cout << "LCS of " << X << " and " << Y << " is " << lcs;
1468            }
1469
1470            ////////////////////////// LCS END //////////////////////////
1471
1472
1473            ////////////////////////// LPS //////////////////////////
1474
1475            //int dp[1005][1005];
```

```
1476            // Returns the length of the longest palindromic subsequence in seq
1477            int lps (string seq, int i, int j) {
1478                // Base Case 1: If there is only 1 character
1479                if (i == j)
1480                    return 1;
1481
1482                // Base Case 2: If there are only 2 characters and both are same
1483                if (seq[i] == seq[j] && i + 1 == j)
1484                    return 2;
1485
1486                if (dp[i][j] != -1) return dp[i][j];
1487
1488                // If the first and last characters match
1489                if (seq[i] == seq[j])
1490                    return dp[i][j] = lps (seq, i + 1, j - 1) + 2;
1491
1492                // If the first and last characters do not match
1493                return dp[i][j] = max (lps (seq, i, j - 1), lps (seq, i + 1, j) );
1494            }
1495
1496
1497            // Returns the length of the longest palindromic subsequence in seq
1498            int lps_it (string str) {
1499                short DP[1001][1001];
1500                memset (DP, 0, sizeof (DP) );
1501                int len = (int) str.length(), i, j;
1502
1503                for (i = 0; i < len; i++) {
1504                    for (j = 0; j + i < len; j++) {
1505                        if (str[j] == str[i + j]) {
1506                            DP[j][j + i] = (short) (DP[j + 1][j + i - 1] +  (i == 0
                                ? 1 : 2) );
1507                        } else {
1508                            DP[j][j + i] = max (DP[j + 1][j + i], DP[j][j + i - 1]);
1509                        }
1510                    }
1511                }
1512
1513                return DP[0][len - 1];
1514            }
1515
1516
1517            // Function return the total palindromic subsequence
1518            int countPS (string str) {
1519                int N = (int) str.length();
1520                // create a 2D array to store the count of palindromic
1521                // subsequence
1522                int cps[N + 1][N + 1];
1523                memset (cps, 0, sizeof (cps) );
1524
1525                // palindromic subsequence of length 1
1526                for (int i = 0; i < N; i++)
1527                    cps[i][i] = 1;
1528
1529                // check subsequence of length L is palindrome or not
1530                for (int L = 2; L <= N; L++) {
1531                    for (int i = 0; i < N; i++) {
```

```
1532                        int k = L + i - 1;
1533
1534                        if (str[i] == str[k])
1535                            cps[i][k] = cps[i][k - 1] + cps[i + 1][k] + 1;
1536                        else
1537                            cps[i][k] = cps[i][k - 1] + cps[i + 1][k] - cps[i + 1][k
                                - 1];
1538                    }
1539                }
1540
1541                // return total palindromic subsequence
1542                return cps[0][N - 1];
1543            }
1544
1545            ////////////////////////// LPS END //////////////////////////
1546
1547
1548
1549            ////////////////////////// LIS //////////////////////////
1550
1551            // Binary search (note boundaries in the caller)
1552            int CeilIndex (std::vector<int> &v, int l, int r, int key) {
1553                while (r - l > 1) {
1554                    int m = l + (r - l) / 2;
1555
1556                    if (v[m] >= key)
1557                        r = m;
1558                    else
1559                        l = m;
1560                }
1561
1562                return r;
1563            }
1564
1565            int LongestIncreasingSubsequenceLength (vector<int> &v) {
1566                if (v.size() == 0)
1567                    return 0;
1568
1569                std::vector<int> tail (v.size(), 0);
1570                int length = 1; // always points empty slot in tail
1571                tail[0] = v[0];
1572
1573                for (size_t i = 1; i < v.size(); i++) {
1574                    if (v[i] < tail[0])
1575                        // new smallest value
1576                        tail[0] = v[i];
1577                    else if (v[i] > tail[length - 1])
1578                        // v[i] extends largest subsequence
1579                        tail[length++] = v[i];
1580                    else
1581                        // v[i] will become end candidate of an existing subsequence or
1582                        // Throw away larger elements in all LIS, to make room for
1583                        // upcoming grater elements than v[i].
1584                        // (and also, v[i] would have already appeared in one of LIS,
1585                        // identify the location and replace it)
1586                        tail[CeilIndex (tail, -1, length - 1, v[i])] = v[i];
1587                }
```

```
1588
1589            return length;
1590        }
1591
1592        ////////////////////////////////// LIS END
            ////////////////////////////////
1593
1594    };
1595
1596    ////////////////////////// IS SUBSECUENCE //////////////////////////
1597
1598
1599    ////////////////////////////// KADANE'S ALGO IN 1D //////////////////////////////
1600
1601    // Implementation of Kadane's algorithm for 1D array. The function
1602    // returns the maximum sum and stores starting and ending indexes of the
1603    // maximum sum subarray at addresses pointed by start and finish pointers
1604    // respectively.
1605    int kadane (int *arr, int *start, int *finish, int n) {
1606        // initialize sum, maxSum and
1607        int sum = 0, maxSum = INT_MIN, i;
1608        // Just some initial value to check for all negative values case
1609        *finish = -1;
1610        // local variable
1611        int local_start = 0;
1612
1613        for (i = 0; i < n; ++i) {
1614            sum += arr[i];
1615
1616            if (sum < 0) {
1617                sum = 0;
1618                local_start = i + 1;
1619            } else if (sum > maxSum) {
1620                maxSum = sum;
1621                *start = local_start;
1622                *finish = i;
1623            }
1624        }
1625
1626        // There is at-least one non-negative number
1627        if (*finish != -1)
1628            return maxSum;
1629
1630        // Special Case: When all numbers in arr[] are negative
1631        maxSum = arr[0];
1632        *start = *finish = 0;
1633
1634        // Find the maximum element in array
1635        for (i = 1; i < n; i++) {
1636            if (arr[i] > maxSum) {
1637                maxSum = arr[i];
1638                *start = *finish = i;
1639            }
1640        }
1641
1642        return maxSum;
1643    }
```

```
1644
1645  ////////////////////////// KADANE'S ALGO IN 1D END ////////////////////////
1646
1647
1648  ////////////////////////////// KADANE'S ALGO IN 2D ////////////////////////////
1649
1650  // The main function that finds maximum sum rectangle in M[][]
1651  // use Kadane's algo in 1D, descripted avobe
1652  int MAT[101][101];
1653  int findMaxSum2D (int *finalTop, int *finalLeft, int *finalBottom,
1654                    int *finalRight, int ROW, int COL) {
1655      // Variables to store the final output
1656      int maxSum = INT_MIN;
1657      int left, right, i;
1658      int temp[ROW], sum, start, finish;
1659
1660      // Set the left column
1661      for (left = 0; left < COL; ++left) {
1662          // Initialize all elements of temp as 0
1663          memset (temp, 0, sizeof (temp) );
1664
1665          // Set the right column for the left column set by outer loop
1666          for (right = left; right < COL; ++right) {
1667              // Calculate sum between current left and right for every row 'i'
1668              for (i = 0; i < ROW; ++i)
1669                  temp[i] += MAT[i][right];
1670
1671              // Find the maximum sum subarray in temp[]. The kadane()
1672              // function also sets values of start and finish.  So 'sum' is
1673              // sum of rectangle between (start, left) and (finish, right)
1674              //  which is the maximum sum with boundary columns strictly as
1675              //  left and right.
1676              sum = kadane (temp, &start, &finish, ROW);
1677
1678              // Compare sum with maximum sum so far. If sum is more, then
1679              // update maxSum and other output values
1680              if (sum > maxSum) {
1681                  maxSum = sum;
1682                  *finalLeft = left;
1683                  *finalRight = right;
1684                  *finalTop = start;
1685                  *finalBottom = finish;
1686              }
1687          }
1688      }
1689
1690      return maxSum;
1691  }
1692
1693  ////////////////////////// KADANE'S ALGO IN 2D END ////////////////////////
1694
1695
1696  ////////////////////////// INTEGER TO ROMAN NUMERAL ////////////////////////
1697
1698  /*
1699   * http://rosettacode.org/wiki/Roman_numerals/Encode#C.2B.2B
1700   *
```

```
1701    * Create a function taking a positive integer as its parameter and returning a
1702    * string containing the Roman numeral representation of that integer. Modern
1703    * Roman numerals are written by expressing each digit separately, starting
1704    * with the left most digit and skipping any digit with a value of zero.
1705    */
1706
1707   string to_roman (int value) {
1708       struct romandata_t {
1709           int value;
1710           char const *numeral;
1711       };
1712       static romandata_t const romandata[] = {
1713           {1000, "M"},
1714           {900, "CM"},
1715           {500, "D"},
1716           {400, "CD"},
1717           {100, "C"},
1718           {90, "XC"},
1719           {50, "L"},
1720           {40, "XL"},
1721           {10, "X"},
1722           {9, "IX"},
1723           {5, "V"},
1724           {4, "IV"},
1725           {1, "I"},
1726           {0, NULL}
1727       }; // end marker
1728       string result;
1729
1730       for (romandata_t const *current = romandata; current->value > 0; ++current) {
1731           while (value >= current->value) {
1732               result += current->numeral;
1733               value  -= current->value;
1734           }
1735       }
1736
1737       return result;
1738   }
1739
1740   ////////////////////// INTEGER TO ROMAN NUMERAL END //////////////////////
1741
1742
1743   ////////////////////////////// SLIDING WINDOW //////////////////////////////
1744   // Returns maximum sum in a subarray of size k.
1745   int maxSum (int arr[], int n, int k) {
1746       // k must be greater
1747       if (n < k)
1748           return -1;
1749
1750       // Compute sum of first window of size k
1751       int max_sum = 0;
1752
1753       for (int i = 0; i < k; i++)
1754           max_sum += arr[i];
1755
1756       // Compute sums of remaining windows by
1757       // removing first element of previous
```

```
1758        // window and adding last element of
1759        // current window.
1760        int window_sum = max_sum;
1761
1762        for (int i = k; i < n; i++) {
1763            window_sum += arr[i] - arr[i - k];
1764            max_sum = max (max_sum, window_sum);
1765        }
1766
1767        return max_sum;
1768    }
1769    ////////////////////////// SLIDING WINDOW END //////////////////////////
1770
1771    ////////////////// MAXIMUM OF ALL SUBARRAYS OF SIZE K //////////////////
1772    // A Dequeue (Double ended queue) based method for printing maixmum element of
1773    // all subarrays of size k
1774    vector<int> maxofallKsubarray (int arr[], int n, int k) {
1775        // Create a Double Ended Queue, Qi that will store indexes of array
1776        // elements.. The queue will store indexes of useful elements in every
1777        // window and it will maintain decreasing order of values from front to
1778        // rear in Qi, i.e., arr[Qi.front[]] to arr[Qi.rear()] are sorted in
1779        // decreasing order
1780        vector<int> v;
1781        std::deque<int>  Qi (k);
1782        /* Process first k (or first window) elements of array */
1783        int i;
1784
1785        for (i = 0; i < k; ++i) {
1786            // For very element, the previous smaller elements are useless so
1787            // remove them from Qi
1788            while ( (!Qi.empty() ) && arr[i] >= arr[Qi.back()])
1789                Qi.pop_back();  // Remove from rear
1790
1791            // Add new element at rear of queue
1792            Qi.push_back (i);
1793        }
1794
1795        // Process rest of the elements, i.e., from arr[k] to arr[n-1]
1796        for ( ; i < n; ++i) {
1797            // The element at the front of the queue is the largest element of
1798            // previous window, so print it
1799            v.push_back (arr[Qi.front()]);
1800
1801            // Remove the elements which are out of this window
1802            while ( (!Qi.empty() ) && Qi.front() <= i - k)
1803                Qi.pop_front();  // Remove from front of queue
1804
1805            // Remove all elements smaller than the currently
1806            // being added element (remove useless elements)
1807            while ( (!Qi.empty() ) && arr[i] >= arr[Qi.back()])
1808                Qi.pop_back();
1809
1810            // Add current element at the rear of Qi
1811            Qi.push_back (i);
1812        }
1813
1814        // Print the maximum element of last window
```

```
1815        v.push_back (arr[Qi.front()]);
1816        return v;
1817    }
1818
1819    ////////////////// MAXIMUM OF ALL SUBARRAYS OF SIZE K //////////////////
1820
1821    ////////////////// MINIMUM OF ALL SUBARRAYS OF SIZE K //////////////////
1822    // A Dequeue (Double ended queue) based method for printing maixmum element of
1823    // all subarrays of size k
1824    vector<int> minofallKsubarray (int arr[], int n, int k) {
1825        // Create a Double Ended Queue, Qi that will store indexes of array
1826        // elements The queue will store indexes of useful elements in every window
1827        // and it will maintain decreasing order of values from front to rear in
1828        // Qi, i.e., arr[Qi.front[]] to arr[Qi.rear()] are sorted in decreasing
1829        // order
1830        vector<int> v;
1831        std::deque<int>  Qi (k);
1832        /* Process first k (or first window) elements of array */
1833        int i;
1834
1835        for (i = 0; i < k; ++i) {
1836            // For very element, the previous smaller elements are useless so
1837            // remove them from Qi
1838            while ( (!Qi.empty() ) && arr[i] <= arr[Qi.back()])
1839                Qi.pop_back();  // Remove from rear
1840
1841            // Add new element at rear of queue
1842            Qi.push_back (i);
1843        }
1844
1845        // Process rest of the elements, i.e., from arr[k] to arr[n-1]
1846        for ( ; i < n; ++i) {
1847            // The element at the front of the queue is the largest element of
1848            // previous window, so print it
1849            v.push_back (arr[Qi.front()]);
1850
1851            // Remove the elements which are out of this window
1852            while ( (!Qi.empty() ) && Qi.front() <= i - k)
1853                Qi.pop_front();  // Remove from front of queue
1854
1855            // Remove all elements smaller than the currently
1856            // being added element (remove useless elements)
1857            while ( (!Qi.empty() ) && arr[i] <= arr[Qi.back()])
1858                Qi.pop_back();
1859
1860            // Add current element at the rear of Qi
1861            Qi.push_back (i);
1862        }
1863
1864        // Print the maximum element of last window
1865        v.push_back (arr[Qi.front()]);
1866        return v;
1867    }
1868    ////////////////// MINIMUM OF ALL SUBARRAYS OF SIZE K //////////////////
1869
1870    ////////////////////////////// FACTORIAL ////////////////////////////////
1871    long long factorial (int n) {
```

```
1872        if (n == 0) return 1;
1873        else return (n * factorial (n - 1) );
1874    }
1875
1876    // // C++ program to find last non-zero digit in n!
1877    // http://www.geeksforgeeks.org/last-non-zero-digit-factorial/
1878    // Initialize values of last non-zero digit of
1879    // numbers from 0 to 9
1880    int dig[] = {1, 1, 2, 6, 4, 2, 2, 4, 2, 8};
1881
1882    int lastNon0Digit (int n) {
1883        if (n < 10)
1884            return dig[n];
1885
1886        // Check whether tens (or second last) digit
1887        // is odd or even
1888        // If n = 375, So n/10 = 37 and (n/10)%10 = 7
1889        // Applying formula for even and odd cases.
1890        if ( ( (n / 10) % 10) % 2 == 0)
1891            return (6 * lastNon0Digit (n / 5) * dig[n % 10]) % 10;
1892        else
1893            return (4 * lastNon0Digit (n / 5) * dig[n % 10]) % 10;
1894    }
1895
1896    // Function to return trailing 0s in factorial of n
1897    // http://www.geeksforgeeks.org/count-trailing-zeroes-factorial-number/
1898    int findTrailingZeros (int  n) {
1899        // Initialize result
1900        int count = 0;
1901
1902        // Keep dividing n by powers of 5 and update count
1903        for (int i = 5; n / i >= 1; i *= 5)
1904            count += n / i;
1905
1906        return count;
1907    }
1908
1909    ////////////////////////////// FACTORIAL //////////////////////////////
1910
1911
1912    ////////////////////////////// PARSING //////////////////////////////
1913
1914    // for single character delimiter
1915    vector<string> split (const string &s, char delim) {
1916        vector<string> result;
1917        stringstream ss (s);
1918        string item;
1919
1920        while (getline (ss, item, delim) ) {
1921            result.push_back (item);
1922        }
1923
1924        return result;
1925    }
1926
1927    // for string delimiter
1928    vector<string> split (string s, string delimiter) {
```

```
1929        size_t pos_start = 0, pos_end, delim_len = delimiter.length();
1930        string token;
1931        vector<string> res;
1932
1933        while ( (pos_end = s.find (delimiter, pos_start) ) != string::npos) {
1934            token = s.substr (pos_start, pos_end - pos_start);
1935            pos_start = pos_end + delim_len;
1936            res.push_back (token);
1937        }
1938
1939        res.push_back (s.substr (pos_start) );
1940        return res;
1941    }
1942
1943    ////////////////////////////// PARSING //////////////////////////////
1944
1945    //////////////////////////// STRING MATHCING ////////////////////////////
1946    // http://www.geeksforgeeks.org/searching-for-patterns-set-2-kmp-algorithm/
1947
1948    // Fills lps[] for given patttern pat[0..M-1]
1949    void computeLPSArray (char *pat, int M, int *lps) {
1950        // length of the previous longest prefix suffix
1951        int len = 0;
1952        lps[0] = 0; // lps[0] is always 0
1953        // the loop calculates lps[i] for i = 1 to M-1
1954        int i = 1;
1955
1956        while (i < M) {
1957            if (pat[i] == pat[len]) {
1958                len++;
1959                lps[i] = len;
1960                i++;
1961            } else { // (pat[i] != pat[len])
1962                // This is tricky. Consider the example.
1963                // AAACAAAA and i = 7. The idea is similar
1964                // to search step.
1965                if (len != 0) {
1966                    len = lps[len - 1];
1967                    // Also, note that we do not increment
1968                    // i here
1969                } else { // if (len == 0)
1970                    lps[i] = 0;
1971                    i++;
1972                }
1973            }
1974        }
1975    }
1976
1977    // Prints occurrences of txt[] in pat[]
1978    void KMPSearch (char *pat, char *txt) {
1979        int M = (int) strlen (pat);
1980        int N = (int) strlen (txt);
1981        // create lps[] that will hold the longest prefix suffix
1982        // values for pattern
1983        int lps[M];
1984        // Preprocess the pattern (calculate lps[] array)
1985        computeLPSArray (pat, M, lps);
```

```
1986        int i = 0;  // index for txt[]
1987        int j  = 0;  // index for pat[]
1988
1989        while (i < N) {
1990            if (pat[j] == txt[i]) {
1991                j++;
1992                i++;
1993            }
1994
1995            if (j == M) {
1996                printf ("Found pattern at index %d \n", i - j);
1997                j = lps[j - 1];
1998            }
1999            // mismatch after j matches
2000            else if (i < N && pat[j] != txt[i]) {
2001                // Do not match lps[0..lps[j-1]] characters,
2002                // they will match anyway
2003                if (j != 0)
2004                    j = lps[j - 1];
2005                else
2006                    i = i + 1;
2007            }
2008        }
2009    }
2010
2011    /////////////////////////// STRING MATHCING ///////////////////////////
2012
2013
2014    /////////////////////////// N QUEEN ///////////////////////////
2015
2016    class N_Queen {
2017            int mat[1000][1000];
2018            int n_queen_board_size = 0;
2019            bool is_poss;
2020
2021            bool isSafe (int row, int col) {
2022                int i, j;
2023
2024                for (i = 0; i < col; i++)
2025                    if (mat[row][i])
2026                        return false;
2027
2028                for (i = row, j = col; i >= 0 && j >= 0; i--, j--)
2029                    if (mat[i][j])
2030                        return false;
2031
2032                for (i = row, j = col; j >= 0 && i < n_queen_board_size; i++, j--)
2033                    if (mat[i][j])
2034                        return false;
2035
2036                return true;
2037            }
2038
2039            bool n_queen (int col) {
2040                if (col >= n_queen_board_size)
2041                    return true;
2042
```

```
2043                for (int i = 0; i < n_queen_board_size; i++) {
2044                    if ( isSafe (i, col) ) {
2045                        mat[i][col] = 1;
2046
2047                        if ( n_queen (col + 1) )
2048                            return true;
2049
2050                        mat[i][col] = 0; // BACKTRACK
2051                    }
2052                }
2053
2054                return false;
2055            }
2056
2057        public:
2058            N_Queen();
2059            N_Queen (int n) {
2060                n_queen_board_size = n;
2061                memset (mat, 0, sizeof (mat) );
2062                is_poss = n_queen (0);
2063            }
2064
2065            bool solve (int n) {
2066                n_queen_board_size = n;
2067                memset (mat, 0, sizeof (mat) );
2068                is_poss = n_queen (0);
2069                return is_poss;
2070            }
2071
2072            bool print (void) {
2073                if (!has_sol() ) {
2074                    cout << "No solution found\n";
2075                    return false;
2076                }
2077
2078                for (int i = 0; i < n_queen_board_size; i++) {
2079                    for (int j = 0; j < n_queen_board_size; j++) {
2080                        cout << mat[i][j] << ' ';
2081                    }
2082
2083                    cout << endl;
2084                }
2085
2086                return true;
2087            }
2088            bool has_sol (void) {
2089                return is_poss;
2090            }
2091
2092    };
2093
2094    ////////////////////////////////// N QUEEN ////////////////////////////////////
2095
2096    ////////////////////////////// TRAVELLING SALESMAN ////////////////////////////
2097    class TSP {
2098        public:
2099            int dist[15][15];
```

```
2100            int dp[ (1 << 12) + 5][12];
2101
2102            int tsp (int n) {
2103                int p, ans;
2104                // Run Floyd–Warshall to remove Triangle-Inequality
2105                // It is not necessary for TSP
2106                //for (k = 0; k < n; k++) {
2107                //    for (i = 0; i < n; i++) {
2108                //        for (j = 0; j < n; j++) {
2109                //            if (i != j && i != k && j != k)
2110                //                dist[i][j] = min(dist[i][k] + dist[k][j],
                    dist[i][j]);
2111                //        }
2112                //    }
2113                //}
2114                memset (dp, -1, sizeof (dp) );
2115                dp[1][0] = 0;
2116
2117                for (int i = 1; i < (1 << n); i++) {
2118                    for (int j = 0; j < n; j++) {
2119                        if (dp[i][j] == -1) continue;
2120
2121                        for (int k = 1; k < n; k++) {
2122                            if ( (i & (1 << k) ) != 0) continue; // check either kth
                            bit of i is 0 : 1
2123
2124                            p = (i | (1 << k) ); // ON kth bit of i and store it p
2125
2126                            if (dp[p][k] == -1) dp[p][k] = dp[i][j] + dist[j][k];
2127
2128                            dp[p][k] = min (dp[p][k], dp[i][j] + dist[j][k]);
2129                        }
2130                    }
2131                }
2132
2133                ans = INF;
2134
2135                for (int i = 1; i < n; i++) {
2136                    if (dp[ (1 << n) - 1][i] > 0) ans = min (ans,
2137                                                            dp[ (1 << n) - 1][i] +
                                                            dist[i][0]);
2138                }
2139
2140                return ans;
2141            }
2142    };
2143    ////////////////////////////// TRAVELLING SALESMAN //////////////////////////
2144
2145
2146    ////////////////////////// CONVEX HULL GRAHAM SCAN //////////////////////////
2147    struct Point {
2148        int x, y;
2149    };
2150
2151    vector<Point> points;
2152
2153    void get_points (int x, int y) {
```

```
2155  }
2156
2157  Point p0;
2158
2159  Point nextToTop (stack<Point> &S) {
2160      Point p = S.top();
2161      S.pop();
2162      Point res = S.top();
2163      S.push (p);
2164      return res;
2165  }
2166
2167  int distSq (Point p1, Point p2) {
2168      return (p1.x - p2.x) * (p1.x - p2.x) +
2169             (p1.y - p2.y) * (p1.y - p2.y);
2170  }
2171
2172  int orientation (Point p, Point q, Point r) {
2173      int val = (q.y - p.y) * (r.x - q.x) -
2174                (q.x - p.x) * (r.y - q.y);
2175
2176      if (val == 0) return 0;
2177
2178      return (val > 0) ? 1 : 2;
2179  }
2180
2181  int compare (const void *vp1, const void *vp2) {
2182      Point *p1 = (Point *) vp1;
2183      Point *p2 = (Point *) vp2;
2184      int o = orientation (p0, *p1, *p2);
2185
2186      if (o == 0)
2187          return (distSq (p0, *p2) >= distSq (p0, *p1) ) ? -1 : 1;
2188
2189      return (o == 2) ? -1 : 1;
2190  }
2191
2192  void convexHull() {
2193      int n = (int) points.size();
2194      int ymin = points[0].y, min = 0;
2195
2196      for (int i = 1; i < n; i++) {
2197          int y = points[i].y;
2198
2199          if ( (y < ymin) || (ymin == y &&
2200                              points[i].x < points[min].x) )
2201              ymin = points[i].y, min = i;
2202      }
2203
2204      swap (points[0], points[min]);
2205      p0 = points[0];
2206      qsort (&points[1], n - 1, sizeof (Point), compare);
2207      int m = 1;
2208
2209      for (int i = 1; i < n; i++) {
2210          while (i < n - 1 && orientation (p0, points[i],
2211                                           points[i + 1]) == 0)
```

```
2212                i++;
2213
2214            points[m] = points[i];
2215            m++;
2216        }
2217
2218        if (m < 3) return;
2219
2220        stack<Point> S;
2221        S.push (points[0]);
2222        S.push (points[1]);
2223        S.push (points[2]);
2224
2225        for (int i = 3; i < m; i++) {
2226            while (orientation (nextToTop (S), S.top(), points[i]) != 2)
2227                S.pop();
2228
2229            S.push (points[i]);
2230        }
2231
2232        while (!S.empty() ) {
2233            Point p = S.top();
2234            cout << "(" << p.x << ", " << p.y << ")" << endl;
2235            S.pop();
2236        }
2237    }
2238
2239    /////////////////////////// CONVEX HULL GRAHAM SCAN ////////////////////////
2240
2241
2242    ////////////////////////////// MAGIC SQUARE ////////////////////////////////
2243
2244    class Magic_Square {
2245        public:
2246            int mat[1000][1000];
2247
2248            void print (int n) {
2249                int rowsum, corner = 0;
2250                vi v (n);
2251                rep (i, n) corner += mat[i][i];
2252                cout << "\nCalculated magic number is: " << (n * n * n + n) / 2 <<
                    endl;
2253                cout << "Magic square of order " << n << ":\n\n";
2254                rep (i, n) {
2255                    rowsum = 0;
2256                    rep (j, n) {
2257                        cout << mat[i][j] << '\t';
2258                        rowsum += mat[i][j];
2259                        v[i] += mat[i][j];
2260                    }
2261                    cout << "|" << rowsum << endl;
2262                }
2263                rep (i, n) cout << "________";
2264                cout << "|_____" << endl;
2265                rep (i, n) cout << v[i] << '\t';
2266                cout << "|" << corner;
2267                cout << endl << endl;
```

```
2269
2270            void magic_single (int n) {
2271                cout << "I have found no pattern for order " << n << " :-(\n";
2272            }
2273
2274            void magic_double (int n) {
2275                int mn = n / 4;
2276
2277                for (int i = 0, revY = n - 1; i < n; i++, revY--) {
2278                    for (int j = 0, revX = n - 1; j < n; j++, revX--) {
2279                        if (j < mn or j >= (n - mn) ) {
2280                            if (i < mn or i >= (n - mn) ) mat[i][j] = (i * n) +  (j
                                + 1);
2281                            else mat[i][j] = (revY * n) + (revX + 1);
2282                        } else {
2283                            if (i >= mn and i < (n - mn) ) mat[i][j] = (i * n) +  (j
                                + 1);
2284                            else mat[i][j] = (revY * n) + (revX + 1);
2285                        }
2286
2287                        //debug3(i, j, mat[i][j]);
2288                    }
2289                }
2290            }
2291
2292            void magic_odd (int n) {
2293                int blocks = (n * n), preX = n / 2, preY = 0;
2294
2295                for (int i = 1, x = n / 2, y = 0; i <= blocks; i++, x++, y--) {
2296                    if (x == n and y == -1) x--, y += 2;
2297                    else if (x == n) x = 0;
2298                    else if (y == -1 ) y = n - 1;
2299
2300                    if (mat[y][x]) x = preX, y = preY + 1;
2301
2302                    mat[y][x] = i;
2303                    preX = x, preY = y;
2304                    //debug3(x, y, mat[y][x]);
2305                }
2306            }
2307    };
2308    ////////////////////////////// MAGIC SQUARE //////////////////////////////
2309
2310
2311    ////////////////////////////// PALINDROMES //////////////////////////////
2312
2313    class Palindrome {
2314
2315        public:
2316            int isPalindrome (int n) {
2317                // Find reverse of n
2318                int rev = 0;
2319
2320                for (int i = n; i > 0; i /= 10)
2321                    rev = rev * 10 + i % 10;
2322
2323                // If n and rev are same, then n is palindrome
```

```
2325            }
2326
2327            // A utility for creating palindrome
2328            int createPalindrome (int input, int b, bool isOdd) {
2329                int n = input;
2330                int palin = input;
2331
2332                // checks if number of digits is odd or even
2333                // if odd then neglect the last digit of input in
2334                // finding reverse as in case of odd number of
2335                // digits middle element occur once
2336                if (isOdd)
2337                    n /= b;
2338
2339                // Creates palindrome by just appending reverse
2340                // of number to itself
2341                while (n > 0) {
2342                    palin = palin * b + (n % b);
2343                    n /= b;
2344                }
2345
2346                return palin;
2347            }
2348
2349            // Fruition to print decimal palindromic number
2350            void generatePaldindromes (int n) {
2351                int number;
2352
2353                // Run two times for odd and even length palindromes
2354                for (int j = 0; j < 2; j++) {
2355                    // Creates palindrome numbers with first half as i.
2356                    // Value of j decided whether we need an odd length
2357                    // of even length palindrome.
2358                    int i = 1;
2359
2360                    while ( (number = createPalindrome (i, 10, j % 2) ) < n) {
2361                        cout << number << " ";
2362                        i++;
2363                    }
2364                }
2365            }
2366    };
2367    ////////////////////////////// PALINDROMES ////////////////////////////
2368
2369    //Find if string ends with another string
2370    //https://stackoverflow.com/a/874160/7829174
2371    bool hasEnding (std::string const &fullString, std::string const &ending) {
2372        if (fullString.length() >= ending.length() ) {
2373            return (0 == fullString.compare (fullString.length() - ending.length(),
2374                                             ending.length(), ending) );
2375        } else {
2376            return false;
2377        }
2378    }
2379    //std::string ending = "nary";
2380    //std::string test1 = "binary";
2381    //std::cout << hasEnding (test1, ending) << std::endl;
```

```
// This will return 1, as binary ends with binary;

int main() {
    __FastIO;
    cout << "Hello World!\n";
    return 0;
}

////////////////////////////////////////////////////////////////////////////

/*************************************************************************
_____________________________BUILT_IN FUNCTIONS_____________________________


-------------------------------------------------------------------------
int sprintf(char *restrict buffer, const char *restrictformat, ...);
This function convert number to string with specified format.
Example:
int aInt = 368;
char str[15];
sprintf(str, "%d", aInt);
cout << str << endl;

int sscanf(const char *str, const char *format, ...)
sscanf (sentence,"%s %*s %d",str,&i);
On success, the function returns the number of items in the argument
list successfully filled. This count can match the expected number of
items or be less (even zero) in the case of a matching failure.
In the case of an input failure before any data could be successfully
interpreted, EOF is returned.
Defined in header <stdio.h>


-------------------------------------------------------------------------
std::cin.getline();
getline can be provided a third argument--a "stop" character. This
character ends getline's input. The character is eaten and the
string is terminated. Example: std::cin.getline(str, 100, '|')
If std::cin.getline() is not provided a "stop" character as a third
argument, it will stop when it reaches a newline.
Example: std::cin.getline(str, 100)


-------------------------------------------------------------------------
std::cin.ignore()
can be called three different ways:
1. No arguments: A single character is taken from the input buffer
and discarded:
std::cin.ignore(); //discard 1 character
2. One argument: The number of characters specified are taken from
the input buffer and discarded:
std::cin.ignore(33); //discard 33 characters
3. Two arguments: discard the number of characters specified, or
discard characters up to and including the specified delimiter
(whichever comes first):
std::cin.ignore(26, '\n');
```

```
2439  //ignore 26 characters or to a newline, whichever comes first
2440
2441
2442
2443  ------------------------------------------------------------------------
2444  Formated I/O
2445  Example: std::cout << std::right << setw(5) << 123 << endl;
2446  //output:"  123"
2447  std::cout.fill('X');
2448  std::cout << setw(2) << one << std::endl;
2449  //output: "X4"
2450  #include <iomanip>
2451
2452  ------------------------------------------------------------------------
2453  Input and output in C++ is type safe and easy for common formats
2454  using cin and cout. The following program listing shows some common
2455  uses:
2456
2457  #include <iostream.h>
2458  #include <iomanip.h>
2459
2460  int main()
2461  {
2462      int n;
2463      float f;
2464      double d;
2465      char s[100];
2466
2467      // input an integer
2468      cin >> n;
2469      // print an integer, no formatting
2470      cout << n << endl;
2471      // print an integer, padded on left with spaces to total 6 chars
2472      cout << setw(6) << n << endl;
2473      // print an integer, padded on right with spaces to total 6 chars
2474      cout << setw(-6) << n << endl;
2475
2476      // input a string (whitespace delineated)
2477      cin >> s;
2478      // print a string, no formatting
2479      cout << s << endl;
2480      // print a string, padded with spaces on left to 20 chars
2481      cout << setw(20) << s << endl;
2482      // print a string, padded with spaces on right to 20 chars
2483      cout << setiosflags(ios::left) << setw(20) << s << endl;
2484
2485      // input a single precision floating point number
2486      cin >> f;
2487      // print a float, default precision is 6 places
2488      cout << setiosflags(ios::fixed) << f << endl;
2489      // input a double precision floating point number
2490      cin >> d;
2491      // print a double, default precision is 6 places
2492      cout << d << endl;
2493      // print a double, 2 places of precision
2494      cout << setprecision(2) << d << endl;
2495      // print a double, 2 places of precision, padded with space to 10
```

```
2496        cout << setw(10) << setprecision(2) << d << endl;
2497    }
2498
2499    Rember that you can combine C routine sprintf and C++ cout. For
2500    example, if sprintf can give the desired formatting, use it to create
2501    the desired string, then output using C++ I/O. It is probably not a
2502    good idea to mix C and C++ input and output routines since they are
2503    buffered routines and may produce undesirable re-ordering of
2504    input/output.
2505
2506
2507    ------------------------------------------------------------------------
2508    strcpy(char* des, char* src) //Copies src into des.
2509
2510
2511    ------------------------------------------------------------------------
2512    double a = 18.12385;
2513    cout << fixed << setprecision(3);
2514    cout << a << endl;
2515    //output: 18.124
2516    #include <iomanip>
2517    std::fixed, std::setprecision()
2518
2519
2520    ------------------------------------------------------------------------
2521    next_permutation(str, str+strlen(str));
2522    return non-zero if next permutation found, otherwise 0
2523        char str[] = "arafat";
2524        if(next_permutation(str, str+strlen(str)))
2525            cout << str << endl;
2526    //output: arafta
2527    #include <algorithm>
2528
2529
2530
2531    Define _USE_MATH_DEFINES before including math.h to expose these macro
2532    definitions for common math constants.  These are placed under an #ifdef
2533    since these commonly-defined names are not part of the C/C++ standards.
2534
2535
2536    Definitions of useful mathematical constants
2537    M_E        - e
2538    M_LOG2E    - log2(e)
2539    M_LOG10E   - log10(e)
2540    M_LN2      - ln(2)
2541    M_LN10     - ln(10)
2542    M_PI       - pi
2543    M_PI_2     - pi/2
2544    M_PI_4     - pi/4
2545    M_1_PI     - 1/pi
2546    M_2_PI     - 2/pi
2547    M_2_SQRTPI - 2/sqrt(pi)
2548    M_SQRT2    - sqrt(2)
2549    M_SQRT1_2  - 1/sqrt(2)
2550
2551
2552    #define M_E        2.71828182845904523536
```

```
2553   #define M_LOG2E    1.44269504088896340736
2554   #define M_LOG10E   0.43429448190325182765l
2555   #define M_LN2      0.69314718055994530941
2556   #define M_LN10     2.30258509299404568402
2557   #define M_PI       3.14159265358979323846
2558   #define M_PI_2     1.57079632679489661923
2559   #define M_PI_4     0.78539816339744830962
2560   #define M_1_PI     0.31830988618379067154
2561   #define M_2_PI     0.63661977236758134308
2562   #define M_2_SQRTPI 1.12837916709551257390
2563   #define M_SQRT2    1.41421356237309504880
2564   #define M_SQRT1_2  0.70710678118654752440
2565
2566    ************************************************************************/
2567
2568
```