|  |  |
| --- | --- |
| **//ORDERED SET**  #include <ext/pb\_ds/tree\_policy.hpp>  #include <ext/pb\_ds/assoc\_container.hpp>  **using** **namespace** \_\_gnu\_pbds**;**  template **<**typename T**>**  **using** indexed\_set **=** tree**<**T**,** null\_type**,** less**<**T**>,** rb\_tree\_tag**,** tree\_order\_statistics\_node\_update**>;** | // PBDS (Policy Based Data Structure)  **// Ordered Set**  // Delete them  //directory change:  //C:\MinGW\lib\gcc\mingw32\6.3.0\include\c++\ext\pb\_ds\detail\resize\_policy  //.hpp er porer number gula delete korte hobe  #include<ext/pb\_ds/assoc\_container.hpp>  #include<ext/pb\_ds/tree\_policy.hpp>  **using** **namespace** \_\_gnu\_pbds**;**  template**<**class T**>** **using** oset **=** tree**<**T**,** null\_type**,** less**<**T**>,** rb\_tree\_tag**,** tree\_order\_statistics\_node\_update**>;**  //oset <ll> s; --> Declare ordered set  //s.order\_of\_key(val) --> index of value val  //\*(s.find\_by\_order(ind)) --> value at index ind |
| const int limit **=** 1e7**+**7**;**  **//Sieve of Eratosthenes**  //TimeComplexity O(nloglogn)  //canbeuseduntil10^9  vector**<**bool**>** is\_prime**(**limit**+**1**,true);**  void sieve\_of\_eratosthenes**(){**  //Finding out the primes in simple way is\_prime[0] = is\_prime[1] = false;  **for(**int i**=**2**;**i**\***i**<=**limit**;++**i**){**  **if** **(**is\_prime**[**i**])** **{**  primes**.**push\_back**(**i**);**  **for(**intj**=**i**\***i**;**j**<=**limit**;**j**+=**i**){**  is\_prime**[**j**]=false;**  **}**  **}**  **}**  **}**  const int mx **=** 1e8 **+** 9**;** //max value of n  const int mxprm **=** 6e6 **+** 9**;** //max number  **//Sieve Eratosthenes Bitset**  int psz **=** 0**;** //count the number of primes  bitset **<**mx**>** mark**;** //to keep track of primes  uint primes**[**mxprm**];** //to store the primes  void sieve**()** **{** //just a prime sieve code  mark**[**0**]** **=** mark**[**1**]** **=** 1**;**  primes**[**psz**++]** **=** 2**;**  int lim **=** sqrt**(**mx **\*** 1.0**)** **+** 2**;**  **for** **(**int i**=**4**;**i**<**mx**;**i**+=**2**)** mark**[**i**]** **=** 1**;**  **for** **(**int i**=**3**;**i**<**mx**;**i**+=**2**)** **{**  **if** **(!**mark**[**i**])** **{**  primes**[**psz**++]** **=** i**;**  **if** **(**i**<=**lim**)**  **for** **(**int j**=**i**\***i**;**j**<**mx**;**j**+=**i**)**  mark**[**j**]** **=** 1**;**  **}**  **}**  **}** | **//Prime Factorization**  //faster process  //TimeComplexity\_O(sqrt(n)/ln(sqrt(n))+log2(n))  vector**<**long long**>** primes\_factors**(**long long n**){**  vector**<**long long**>** factors**;**  int root **=** sqrt**(**n**);**  **for(**int i**=**0**;**i**<(**int**)**primes**.**size**()** **&&** primes**[**i**]<=**root**;++**i**){**  **if** **(**is\_prime**[**n**])** **{**  **break;**  **}**  **if(**n**%**primes**[**i**]==**0**){**  **while(**n**%**primes**[**i**]==**0**){** //log2(n)  n **/=** primes**[**i**];**  factors**.**push\_back**(**primes**[**i**]);**  **}**  root**=**sqrt**(**n**);**  **}**  **}**  **if(**n**!=**1**){**  factors**.**push\_back**(**n**);**  **}**  **return** factors**;**  **}** |
| //Returns nCr%p using Fermat's  //little theorem.  unsigned long long nCrModPFermat  **(**unsigned long long n**,**int r**,**int p**)**  **{**  **if(**n**<**r**)** **return** 0**;**  **if(**r**==**0**)return** 1**;**  unsigned long long fac**[**n**+**1**];**  fac**[**0**]** **=** 1**;**  **for(**int i**=**1**;**i**<=**n**;**i**++)**  fac**[**i**]=(**fac**[**i**-**1**]\***i**)%**p**;**  **return** **(**fac**[**n**]\***modInverse**(**fac**[**r**],**p**)%**p  **\***modInverse**(**fac**[**n**-**r**],**p**)%**p**)%**p**;**  **}**  int main**()**  **{**  int n**=**10**,**r**=**2**,**p**=**13**;**  cout **<<** "ValueofnCr%p is"  **<<** nCrModPFermat**(**n**,**r**,**p**);**  **}** | //A modular inverse based solution to  //compute nCr % p  /\* Iterative Function to calculate (x^y)%p in O(log y) \*/  unsigned long long power  **(**unsigned long long x**,**int y**,**int p**)**  **{**  unsigned long long res**=**1**;**  x **=** x **%** p**;**  **while(**y**>**0**){**  **if(**y**&**1**)** res **=** **(**res**\***x**)%**p**;**  y **=** y**>>**1**;**//y=y/2  x **=** **(**x **\*** x**)** **%** p**;**  **}**  **return** res**;**  **}**  //Returnsn^(-1)modp  unsigned long long modInverse**(**unsigned long long n**,**int p**)**  **{**  **return** power**(**n**,**p**-**2**,**p**);**  **}** |
| const int maxn**=(**int**)**1e5**+**7**;**  int phi**[**maxn**];**  //TimeComplexity-O(nloglogn)  **// EulerTotient**  void phi\_1\_to\_n**()** **{**  **for(**int i**=**0**;**i**<=**maxn**;++**i**){**  phi**[**i**]=**i**;**  **}**  **for(**int i**=**2**;**i**<=**maxn**;++**i**){**  **if** **(**phi**[**i**]** **==** i**)** **{**  **for(**int j**=**i**;**j**<=**maxn**;**j**+=**i**){**  phi**[**j**]-=**phi**[**j**]/**i**;**  **}**  **}**  **}**  **}**  **//sum of coprimes until n**  int sum\_of\_coprimes\_untill\_n**(**int n**){**  **return** **(**phi**[**n**]/**2**)** **\*** n**;**  **}**  int main**()** **{**  ios\_base**::**sync\_with\_stdio**(false),**  cin**.**tie**(nullptr);**  phi\_1\_to\_n**();**  int n**;** cin **>>** n**;**  **for(**int i**=**2**;**i**<**13**;++**i**){**  cout **<<** phi**[**i**]** **<<**''**;**  **}**  cout **<<** '\n'**;**  cout **<<** sum\_of\_coprimes\_until\_n**(**n**)** **<<** '\n'**;**  **}** | void mat\_mul**(**vector**<**vector**<**ll**>> &**mat1**,** vector**<**vector**<**ll**>>** **&**mat2**){**  vector**<**vector**<**ll**>>** newmat**(**2**,**  vector**<**ll**>(**2**,** 0**));**  **for(**ll i**=**0**;**i**<**2**;**i**++)**  **for(**ll j**=**0**;**j**<**2**;**j**++)**  **for(**ll k**=**0**;**k**<**2**;**k**++)**  newmat**[**i**][**j**]** **+=** mat1**[**i**][**k**]\***mat2**[**k**][**j**];**    mat1 **=** newmat**;**  **}**  ll fib**(**ll n**){**  **if(**n**==**1**)** **return** 0**;**  **if(**n**==**2**)** **return** 1**;**  **if(**n**==**3**)** **return** 1**;**  vector**<**vector**<**ll**>>** resmat**,** mat**;**  resmat **=** mat **=** **{{**0**,** 1**},** **{**1**,** 1**}};**  ll i**;**  n**-=**3**;**  **for(**i**=**0**;(**1ll**<<**i**)<=**n**;**i**++){**  **if(**n**&(**1ll**<<**i**))** mat\_mul**(**resmat**,** mat**);**  mat\_mul**(**mat**,** mat**);**  //cout << mat[0][0] << mat[0][1] << mat[1][0] << mat[1][1];  **}**  **return** resmat**[**1**][**1**];**  **}** |
| **//EEGCD AND Linear Diophantine**  ll gcd**(**ll a**,** ll b**,** ll**&** x**,** ll**&** y**)** **{**  **if** **(**b **==** 0**)** **{**  x **=** 1**;**  y **=** 0**;**  **return** a**;**  **}**  ll x1**,** y1**;**  ll d **=** gcd**(**b**,** a **%** b**,** x1**,** y1**);**  x **=** y1**;**  y **=** x1 **-** y1 **\*** **(**a **/** b**);**  **return** d**;**  **}**  void solve**(**ll cs**){**  ll j**,** i**,** p**,** q**,** a**,** b**,** c**,** m**,** n**,** k**,** g**,** mn **=** 0**,** mx **=** 1e10**;**  cin **>>** n**;**  **while(**n**--){**  cin **>>** a **>>** b**;**  ll x**,** y**;**  ll g **=** gcd**(**abs**(**a**),** abs**(**b**),** x**,** y**);**  **if** **(**a **<** 0**)** x **=** **-**x**;**  **if** **(**b **<** 0**)** y **=** **-**y**;**  cout **<<** g **<<** " " **<<** x **<<** " " **<<** y**;**  cout **<<** " => "**;**  // linear diophantine ax + by = c .. q = c/gcd(a,b)  double c1**;**  cin **>>** c1**;**  double q **=** c1**/**g**;**  cout **<<** "a" **<<** x**\***q **<<** " b" **<<** y**\***q **<<** endl**;**  **}**  cout **<<** endl**;**  **}** | vector**<**ll**>** v**[**100005**];**  ll vis**[**100005**],** d**[**100005**];**  void bfs**(**ll x**){**  ll n**;**  queue**<**ll**>** q**;**  **for(**ll i**=**0**;**i**<**100005**;**i**++)** vis**[**i**]** **=** 0**;**  q**.**push**(**x**);**  d**[**x**]** **=** 0**;**  **while(!**q**.**empty**()){**  x **=** q**.**front**();**  vis**[**x**]** **=** 1**;**  q**.**pop**();**  **for(**auto xx **:** v**[**x**]){**  **if(**vis**[**xx**]==**0**){**  vis**[**xx**]** **=** 1**;**  q**.**push**(**xx**);**  d**[**xx**]** **=** d**[**x**]** **+** 1**;**  **}**  **}**  **}**  **}** |
| vector**<**ll**>** v**[**1000006**];**  void dfs**(**ll i**,** vector**<**ll**>** **&**col**,**  vector**<**ll**>** **&**d**,** vector**<**ll**>** **&**f**,**  vector**<**ll**>** **&**par**,** ll **&**time**)**  **{**  col**[**i**]** **=** 1**;**  d**[**i**]** **=** **++**time**;**  **for(**auto x **:** v**[**i**]){**  **if(**col**[**x**]==**0**)** **{**  par**[**x**]** **=** i**;**  dfs**(**x**,** col**,** d**,** f**,** par**,** time**);**  **}**  **}**  col**[**i**]** **=** 2**;**  f**[**i**]** **=** **++**time**;**  **}** | vector**<**vector**<**pair**<**ll**,** ll**>>>** v**;**  map**<**ll**,** ll**>** dis**;**  ll dijkstra**(**ll i**){**  priority\_queue**<**pair**<**ll**,** ll**>,**  vector**<**pair**<**ll**,** ll**>>,** greater**<**pair**<**ll**,** ll**>>>** pq**;**  dis**[**i**]** **=** 0**;**  pair**<**ll**,** ll**>** pi**;**  pq**.**push**({**0**,** i**});**  **while(!**pq**.**empty**()){**  pi **=** pq**.**top**();**  pq**.**pop**();**  ll u **=** pi**.**second**;**  **for(**auto x **:** v**[**u**]){**  **if(**dis**[**u**]+**x**.**second**<**dis**[**x**.**first**]){**  dis**[**x**.**first**]=**dis**[**u**]+**x**.**second**;**  pq**.**push**({**dis**[**x**.**first**],**x**.**first**});**  **}**  **}**  **}**  **}** |
| **// Bellman-Ford Algorithm**  // Better than Dijkstra if there are negative weights, edges and cycles  // takes three elements in the edges  vector**<**vector**<**int**>>** edge**;**  void bellman\_ford**(**int node**)** **{**  // firstly, every distance is infinity  vector**<**int**>** dist**(**n**,** inf**);**  // source nodes distance to itself is 0  dist**[**node**]** **=** 0**;**  // checking for the shortest path distance  **for** **(**int i **=** 0**;** i **<** n **-** 1**;** **++**i**)** **{**  **for** **(**auto**&** x**:** edge**)** **{**  int u **=** x**[**0**],** v **=** x**[**1**],** w **=** x**[**2**];**  dist**[**v**]** **=** min**(**dist**[**v**],** w **+** dist**[**u**]);**  **}**  **}**  int ok **=** 0**;**  // checking for negative cycles  **for** **(**auto**&** x**:** edge**)** **{**  int u **=** x**[**0**],** v **=** x**[**1**],** w **=** x**[**2**];**  **if** **(**dist**[**u**]** **!=** inf **&&** dist**[**u**]** **+** w **<** dist**[**v**])**  **{**  ok **=** 1**;**  **break;**  **}**  **}**  **if** **(**ok**)** **{**  cout **<<** "Negative Cycle Found\n"**;**  **}**  **else** **{**  **for** **(**int i **=** 1**;** i **<=** n**;** **++**i**)** **{**  cout **<<** dist**[**i**]** **<<** ' '**;**  **}**  cout **<<** '\n'**;**  **}**  // if there is a negative cycle, then the shortest path cannot be found  // else print the answer  **}** | vector**<**pair**<**ll**,** ll**>>** v**[**1000006**];**  **// Prim MST**  double primmst**(**ll i**,** ll n**){**  priority\_queue**<**pair**<**ll**,** ll**>,** vector**<**pair**<**ll**,** ll**>>,** greater**<**pair**<**ll**,** ll**>>>** pq**;**  vector**<**ll**>** key**(**n**+**1**,** 1e9**);**  vector**<**ll**>** par**(**n**+**1**,** **-**1**);**  vector**<**bool**>** inmst**(**n**+**1**,** **false);**  ll src **=** 1**,** tot **=** 0**;**  key**[**src**]** **=** 0**;**  pq**.**push**({**key**[**src**],** src**});**  **while(!**pq**.**empty**()){**  pair**<**ll**,** ll**>** pi **=** pq**.**top**();**  pq**.**pop**();**  **if(**inmst**[**pi**.**se**])** **continue;**  inmst**[**pi**.**se**]** **=** **true;**  tot **+=** pi**.**fi**;**  **for(**auto x **:** v**[**pi**.**se**]){**  **if(!**inmst**[**x**.**fi**]** **and** key**[**x**.**fi**]>**x**.**se**){**  key**[**x**.**fi**]** **=** x**.**se**;**  pq**.**push**({**key**[**x**.**fi**],** x**.**fi**});**  par**[**x**.**fi**]** **=** pi**.**se**;**  **}**  **}**  **}**    **for(**i**=**2**;**i**<**n**+**1**;**i**++)**  cout **<<** par**[**i**]** **<<** " " **<<** i **<<** endl**;**  **return** tot**;**  **}** |
| **// Kruskal (Minimum Spanning Tree)**  // Time complexity O(E log E)  struct DSU **{**  vector**<**int**>** par**,** rnk**,** size**;** int c**;**  DSU**(**int n**)** **:** par**(**n **+** 1**),** rnk**(**n **+** 1**,**0**),** size**(**n **+** 1**,**1**),** c**(**n**)** **{**  **for** **(**int i **=** 1**;** i **<=** n**;** **++**i**)** par**[**i**]** **=** i**;**  **}**  int find**(**int i**)** **{** **return** **(**par**[**i**]** **==** i **?** i **:** **(**par**[**i**]** **=** find**(**par**[**i**])));** **}**  bool same**(**int i**,** int j**)** **{** **return** find**(**i**)** **==** find**(**j**);** **}**  int get\_size**(**int i**)** **{** **return** size**[**find**(**i**)];** **}**  int count**()** **{** **return** c**;** **}** //connected components  // Path compression  // O(1)  int merge**(**int i**,** int j**)** **{**  **if** **((**i **=** find**(**i**))** **==** **(**j **=** find**(**j**)))** **return** **-**1**;** **else** **--**c**;**  **if** **(**rnk**[**i**]** **>** rnk**[**j**])** swap**(**i**,** j**);**  par**[**i**]** **=** j**;** size**[**j**]** **+=** size**[**i**];**  **if** **(**rnk**[**i**]** **==** rnk**[**j**])** rnk**[**j**]++;**  **return** j**;**  **}**  **};**  int main**()** **{**  ios\_base**::**sync\_with\_stdio**(**0**),** cin**.**tie**(**0**);**  int n**,** m**;**  cin **>>** n **>>** m**;**  vector**<**array**<**int**,** 3**>>** edges**;**  **for** **(**int i **=** 1**;** i **<=** m**;** **++**i**)** **{**  int u**,** v**,** w**;**  cin **>>** u **>>** v **>>** w**;**  edges**.**push\_back**({**w**,** u**,** v**});**  **}**  sort**(**edges**.**begin**(),** edges**.**end**());**  long long ans **=** 0**,** cnt\_edges **=** 0**;**  DSU dsu**(**n**);**  **for** **(**auto**&** x**:** edges**)** **{**  int u **=** x**[**1**],** v **=** x**[**2**],** w **=** x**[**0**];**  **if** **(**dsu**.**same**(**u**,** v**))** **{**  **continue;**  **}**  ans **+=** w**;**  dsu**.**merge**(**u**,** v**);**  **++**cnt\_edges**;**  **}**  **if** **(**ans **>=** 0 **&&** cnt\_edges **==** n **-** 1**)** **{**  cout **<<** ans **<<** '\n'**;**  **}**  **else** **{**  cout **<<** "IMPOSSIBLE\n"**;**  **}**  **return** 0**;**  **}** | **//LCA using sparse table**  //Complexity: O(NlgN,lgN)  int L**[**mx**];** //লেভেল  int P**[**mx**][**22**];** //স্পার্স টেবিল  int T**[**mx**];** //প্যারেন্ট  vector**<**int**>**g**[**mx**];**  void dfs**(**int from**,**int u**,**int dep**){**  T**[**u**]=**from**;**  L**[**u**]=**dep**;**  **for(**int i**=**0**;**i**<(**int**)**g**[**u**].**size**();**i**++){**  int v**=**g**[**u**][**i**];**  **if(**v**==**from**)** **continue;**  dfs**(**u**,**v**,**dep**+**1**);**  **}**  **}**  int lca\_query**(**int N**,** int p**,** int q**)** //N=নোড সংখ্যা  **{**  int tmp**,** log**,** i**;**  **if** **(**L**[**p**]** **<** L**[**q**])** swap**(**p**,** q**)**  log**=**1**;**  **while(**1**)** **{**  int next**=**log**+**1**;**  **if((**1**<<**next**)>**L**[**p**])break;**  log**++;**  **}**  **for** **(**i **=** log**;** i **>=** 0**;** i**--)**  **if** **(**L**[**p**]** **-** **(**1 **<<** i**)** **>=** L**[**q**])**  p **=** P**[**p**][**i**];**  **if** **(**p **==** q**)**  **return** p**;**  **for** **(**i **=** log**;** i **>=** 0**;** i**--)**  **if** **(**P**[**p**][**i**]** **!=** **-**1 **&&** P**[**p**][**i**]** **!=** P**[**q**][**i**])**  p **=** P**[**p**][**i**],** q **=** P**[**q**][**i**];**  **return** T**[**p**];**  **}**  void lca\_init**(**int N**){**  memset **(**P**,-**1**,sizeof(**P**));**  int i**,** j**;**  **for** **(**i **=** 0**;** i **<** N**;** i**++)**  P**[**i**][**0**]** **=** T**[**i**];**  **for** **(**j **=** 1**;** 1 **<<** j **<** N**;** j**++)**  **for** **(**i **=** 0**;** i **<** N**;** i**++)**  **if** **(**P**[**i**][**j **-** 1**]** **!=** **-**1**)**  P**[**i**][**j**]** **=** P**[**P**[**i**][**j **-** 1**]][**j **-** 1**];**  **}**  int main**(**void**)** **{**  g**[**0**].**pb**(**1**);**  g**[**0**].**pb**(**2**);**  g**[**2**].**pb**(**3**);**  g**[**2**].**pb**(**4**);**  dfs**(**0**,** 0**,** 0**);**  lca\_init**(**5**);**  printf**(**"%d\n"**,** lca\_query**(**5**,**3**,**4**)** **);**  **return** 0**;**  **}** |
| **// SCC**  procedure DFS**(**G**,** u**):**  color**[**u**]** ← GREY  **for** all edges from u to v in G**.**adjacentEdges**(**u**)** **do**  **if** color**[**v**]=**WHITE  DFS**(**G**,**v**)**  end **if**  end **for**  stk**.**add**(**source**)**  **return**  procedure DFS2**(**R**,**u**,** mark**)**  components**[**mark**].**add**(**u**)** //save the nodes of the new component  visited**[**u**]** ← **true**  **for** all edges from u to v in R**.**adjacentEdges**(**u**)** **do**  **if** visited**[**v**]** ← **false**  DFS2**(**R**,**v**,** mark**)**  end **if**  end **for**  **return**  procedure findSCC**(**G**):**  stk ← an empty stack  visited**[]** ← null  color**[]** ← null  components**[]** ← null  mark**=**0  **for** each u in G  **if** color**[**u**]=**WHITE  DFS**(**G**,**u**)**  end **if**  end **for**  R**=**reverseEdges**(**G**)**  **while** stk **not** empty  u**=**stk**.**removeTop**()**  **if** visited**[**u**]=false**  mark**=**mark**+**1 //A new component found, it will be identified by ‘mark’  DFS2**(**R**,**u**,**mark**)**  end **if**  end **for**  **return** components | vector**<**ll**>** v**[**1000000**];**  **// toposort**  void toposort**(**ll i**,** vector**<**bool**>** **&**vis**,** stack**<**ll**>** **&**st**){**  vis**[**i**]** **=** **true;**  **for(**auto x **:** v**[**i**]){**  **if(!**vis**[**x**])** toposort**(**x**,** vis**,** st**);**  **}**  st**.**push**(**i**);**  **}**  **//Floyd Warshal**  **for** **(**int k **=** 0**;** k **<** n**;** **++**k**)** **{**  **for** **(**int i **=** 0**;** i **<** n**;** **++**i**)** **{**  **for** **(**int j **=** 0**;** j **<** n**;** **++**j**)** **{**  **if** **(**d**[**i**][**k**]** **<** INF **&&** d**[**k**][**j**]** **<** INF**)**  d**[**i**][**j**]** **=** min**(**d**[**i**][**j**],**  d**[**i**][**k**]** **+** d**[**k**][**j**]);**  **}**  **}**  **}** |
| **// Sparse Table**  // Function to build the sparse table  vector**<**vector**<**ll**>>** buildSparseTable**(**  vector**<**ll**>&** arr**)** **{**  ll n **=** arr**.**size**();**  ll k **=** log2**(**n**)** **+** 1**;**  vector**<**vector**<**ll**>>**  sparseTable**(**n**,** vector**<**ll**>(**k**,** **-**1e9**));**  // Initialize sparse table for range with length 1  **for** **(**ll i **=** 0**;** i **<** n**;** i**++)** **{**  sparseTable**[**i**][**0**]** **=** arr**[**i**];**  **}**    // Build sparse table  **for** **(**ll j **=** 1**;** **(**1LL **<<** j**)** **<=** n**;** j**++)** **{**  **for** **(**ll i **=** 0**;** **(**i **+** **(**1LL **<<** j**)** **-** 1**)** **<** n**;** i**++)**  **{**  sparseTable**[**i**][**j**]** **=** max**(**sparseTable**[**i**][**j**-**1**],** sparseTable**[**i **+** **(**1LL **<<** **(**j **-** 1**))][**j **-** 1**]);**  **}**  **}**  **return** sparseTable**;**  **}**  // Function to query the maximum value in a range  ll queryMax**(**vector**<**vector**<**ll**>>&** sparseTable**,** ll l**,** ll r**)** **{**  ll k **=** log2**(**r **-** l **+** 1**);**  **return** max**(**sparseTable**[**l**][**k**],** sparseTable**[**r **-** **(**1LL **<<** k**)** **+** 1**][**k**]);**  **}** | **//Articulation Point**  vector**<**ll**>** parent**,** v**[**200006**],** dis**,** low**;**  vector**<**bool**>** vis**,** arti\_point**;**  ll t **=** 0**,** ans **=** 0**,** root **=** 1**;**  void dfs**(**ll i**){**  vis**[**i**]** **=** **true;**  low**[**i**]** **=** dis**[**i**]** **=** t**++;**  **for(**auto x **:** v**[**i**]){**  **if(**vis**[**x**]==false){**  parent**[**x**]** **=** i**;**  dfs**(**x**);**  **if(**root**==**i**){**  **if(!**arti\_point**[**i**]** **and** dis**[**i**]<**low**[**x**]** **and** v**[**i**].**size**()>**1**){**  arti\_point**[**i**]** **=** **true;**  **}**  **}**  **else** **if(**dis**[**i**]<=**low**[**x**]){**  arti\_point**[**i**]** **=** **true;**  **}**  **else** low**[**i**]** **=** min**(**low**[**i**],** low**[**x**]);**  **}**  **else** **if(**parent**[**i**]!=**x**){**  low**[**i**]** **=** min**(**low**[**i**],** dis**[**x**]);**  **}**  **}**  **}**  void solve**(**ll cs**){**  ll n**,** m**,** a**=**0**,** x**,** y**,** k**,** b**=**0**,** j**,** i**,** c**,** q**,** mn **=** 1e12**,** mx**,** mod **=** 998244353**;**  cin **>>** n **>>** m**;**  **for(**i**=**0**;**i**<=**n**;**i**++)** v**[**i**].**clear**();**  vis **=** vector**<**bool**>** **(**n**+**1**,** 0**);**  parent **=** vector**<**ll**>** **(**n**+**1**,** 0**);**  dis **=** vector**<**ll**>** **(**n**+**1**,** 0**);**  low **=** vector**<**ll**>** **(**n**+**1**,** 0**);**  arti\_point **=** vector**<**bool**>** **(**n**+**1**,** **false);**  **while(**m**--){**  cin **>>** a **>>** b**;**  v**[**a**].**pb**(**b**);**  v**[**b**].**pb**(**a**);**  **}**  ans **=** 0**;**  dfs**(**1**);**  **for(**i**=**1**;**i**<=**n**;**i**++)** **{**  cout **<<** arti\_point**[**i**]** **<<** " "**;**  **if(**arti\_point**[**i**])** ans**++;**  **}**  cout **<<** ans **<<** endl**;**  **}** |
| vector**<**ll**>** par**;**  **// DSU**  ll fd**(**ll r**){**  **if(**r**==**par**[**r**])** **return** r**;**  par**[**r**]** **=** fd**(**par**[**r**]);**  **return** par**[**r**];**  **}**  void uni**(**ll a**,** ll b**){**  ll u **=** fd**(**a**);**  ll v **=** fd**(**b**);**  **if(**u**==**v**){**  cout **<<** "They are already friends" **<<** endl**;**  **}**  **else{**  par**[**u**]** **=** v**;**  **}**  **}**  **// BIT**  int BIT**[**1000**],** a**[**1000**],** n**;**  void update**(**int x**,** int val**)**  **{**  **for(;** x **<=** n**;** x **+=** x**&-**x**)**  BIT**[**x**]** **+=** val**;**  **}**  int query**(**int x**)**  **{**  int sum **=** 0**;**  **for(;** x **>** 0**;** x **-=** x**&-**x**)**  sum **+=** BIT**[**x**];**  **return** sum**;**  **}**  int main**()**  **{**  scanf**(**“**%**d”**,** **&**n**);**  int i**;**  **for(**i **=** 1**;** i **<=** n**;** i**++)**  **{**  scanf**(**“**%**d”**,** **&**a**[**i**]);**  update**(**i**,** a**[**i**]);**  **}**  printf**(**“sum of first 10 elements is **%**d\n”**,** query**(**10**));**  printf**(**“sum of all elements in range **[**2**,** 7**]** is **%**d\n”**,** query**(**7**)** – query**(**2**-**1**));**  **return** 0**;**  **}** | **/\* Prefix Trie \*/**  struct node**{**  bool endmark**;**  node **\***next**[**26**+**1**];**  node**(){**  **for(**ll i**=**0**;**i**<**26**;**i**++)** next**[**i**]** **=** **NULL;**  endmark **=** **false;**  **}**  **};**  node **\***root**;**  void insert**(**string s**){**  ll n **=** s**.**size**();**  node **\***curr **=** root**;**  **for(**ll i**=**0**;**i**<**n**;**i**++){**  **if(**curr**->**next**[**s**[**i**]-**'a'**]==NULL)**  curr**->**next**[**s**[**i**]-**'a'**]** **=** **new** node**();**  curr **=** curr**->**next**[**s**[**i**]-**'a'**];**  **}**  curr**->**endmark **=** 1**;**  **}**  bool search**(**string s**){**  ll n **=** s**.**size**();**  node **\***curr **=** root**;**  **for(**ll i**=**0**;**i**<**n**;**i**++){**  **if(**curr**->**next**[**s**[**i**]-**'a'**]==NULL)**  **return** **false;**  curr **=** curr**->**next**[**s**[**i**]-**'a'**];**  **}**  **return** curr**->**endmark**;**  **}**  void del**(**node**\*** curr**){**  **for(**ll i**=**0**;**i**<**26**;**i**++){**  **if(**curr**->**next**[**i**]!=NULL)**  del**(**curr**->**next**[**i**]);**  **}**  **delete** **(**curr**);**  **}**  int main**(){**  ios\_base**::**sync\_with\_stdio**(false);** cin**.**tie**(NULL);** cout**.**tie**(NULL);**  //seive(1e6+2);  root **=** **new** node**();**  ll i**,** n**;**  cin **>>** n**;**  **for(**i**=**0**;**i**<**n**;**i**++){**  string s**;**  cin **>>** s**;**  insert**(**s**);**  **}**  ll q**;**  cin **>>** q**;**  **while(**q**--){**  string s**;**  cin **>>** s**;**  **if(**search**(**s**))** cout **<<** "Found" **<<** endl**;**  **else** cout **<<** "Not Found" **<<** endl**;**  **}**  del**(**root**);**  **}** |
| struct node**{**  ll val**,** prop**;**  **};**  **//SEGMENT Tree**  vector**<**node**>** seg**(**1000001**);**  vector**<**ll**>** arr**;**  void init**(**ll node**,** ll l**,** ll r**){**  **if(**l**==**r**){**  seg**[**node**].**val **=** arr**[**l**];**  seg**[**node**].**prop **=** 0**;**  **return;**  **}**  ll mid **=** **(**l**+**r**)/**2**;**  init**(**2**\***node**,** l**,** mid**);**  init**(**2**\***node**+**1**,** mid**+**1**,** r**);**  seg**[**node**].**val **=** seg**[**2**\***node**].**val **+**  seg**[**2**\***node**+**1**].**val**;**  seg**[**node**].**prop **=** 0**;**  **}**  ll query**(**ll node**,** ll l**,** ll r**,** ll i**,** ll j**,** ll carry **=** 0**){**  **if(**r**<**i **or** l**>**j**)** **return** 0**;**  **if(**i**<=**l **and** r**<=**j**)** **return** seg**[**node**].**val **+**  carry**\*(**r**-**l**+**1**);**  ll mid **=** **(**l**+**r**)/**2**;**  ll x **=** query**(**2**\***node**,** l**,** mid**,** i**,** j**,**  carry**+**seg**[**node**].**prop**);**  ll y **=** query**(**2**\***node**+**1**,** mid**+**1**,** r**,** i**,** j**,**  carry**+**seg**[**node**].**prop**);**  **return** x**+**y**;**  **}**  void update**(**ll node**,** ll l**,** ll r**,** ll i**,** ll j**,** ll k**){**  **if(**i**<=**l **and** r**<=**j**){**  seg**[**node**].**prop **=** k**;**  seg**[**node**].**val **+=** **(**r**-**l**+**1**)\***k**;**  **return;**  **}**  **if(**j**<**l **or** i**>**r**)** **return;**  ll mid **=** **(**l**+**r**)/**2**;**  update**(**2**\***node**,** l**,** mid**,** i**,** j**,** k**);**  update**(**2**\***node**+**1**,** mid**+**1**,** r**,** i**,** j**,** k**);**  seg**[**node**].**val **=** seg**[**2**\***node**].**val **+**  seg**[**2**\***node**+**1**].**val **+** **(**r**-**l**+**1**)\***seg**[**node**].**prop**;**  **}**  int main**(){**  ll i**,** n**,** j**,** k**;**  cin **>>** n**;**  arr **=** vector**<**ll**>(**n**);**  **for(**auto **&**x **:** arr**)** cin **>>** x**;**  init**(**1**,** 0**,** n**-**1**);**  k **=** query**(**1**,** 0**,** n**-**1**,** 0**,** 6**);**  cout **<<** k **<<** endl**;**  update**(**1**,** 0**,** n**-**1**,** 4**,** 4**,** 10**);**  k **=** query**(**1**,** 0**,** n**-**1**,** 0**,** 6**);**  cout **<<** k **<<** endl**;**  **}** | template**<**class T**>**  struct segtree **{**  int n**;**  vector**<**T**>** tree**;**  vector**<**T**>** lazy**;**  segtree**(**int len**)** **{**  tree**.**resize**(**4 **\*** len**,** 0**);**  lazy**.**resize**(**4 **\*** len**,** 0**);**  n **=** len**;**  **}**  // change combine and push function  T combine**(**T x**,** T y**)** **{**  **return** x **+** y**;**  **}**  void push**(**int at**,** int l**,** int r**)** **{**  **if** **(**lazy**[**at**]** **==** 0**)** **return;**  tree**[**at**]** **+=** lazy**[**at**]** **\*** **(**r **-** l **+** 1**);**  **if** **(**l **!=** r**)** lazy**[**at **<<** 1**]** **+=** lazy**[**at**];**  **if** **(**l **!=** r**)**  lazy**[**at **<<** 1 **|** 1**]** **+=** lazy**[**at**];**  lazy**[**at**]** **=** 0**;**  **}**  void build**(**vector**<**T**>** **&**arr**,** int at**,** int l**,**  int r**)** **{**  **if** **(**l **==** r**)** **{**  tree**[**at**]** **=** arr**[**l**];**  **return** **;**  **}**  int m **=** **(**l **+** r**)** **>>** 1**;**  build**(**arr**,** at **<<** 1**,** l**,** m**);**  build**(**arr**,** at **<<** 1 **|** 1**,** m **+** 1**,** r**);**  tree**[**at**]** **=** combine**(**tree**[**at **<<** 1**],**  tree**[**at **<<** 1 **|** 1**]);**  **}**  void Build**(**vector**<**T**>** **&**arr**){** build**(**arr**,** 1**,** 0**,** n **-** 1**);** **}** // Use this  void update**(**int at**,** int l**,** int r**,** int L**,** int R**,** T val**)** **{**  push**(**at**,** l**,** r**);**  **if** **(**r **<** L **||** R **<** l**)** **return;**  **if** **(**L **<=** l **&&** r **<=** R**)** **{**  lazy**[**at**]** **=** val**;**  push**(**at**,**l**,**r**);**  **return;**  **}**  int m **=** **(**l **+** r**)** **>>** 1**;**  update**(**at **<<** 1**,** l**,** m**,** L**,** R**,** val**);**  update**(**at **<<** 1 **|** 1**,** m **+** 1**,** r**,** L**,** R**,** val**);**  tree**[**at**]** **=** combine**(**tree**[**at **<<** 1**],**  tree**[**at **<<** 1 **|** 1**]);**  **}**  void Update**(**int l**,** int r**,** T val**)** **{** update**(**1**,** 0**,** n **-** 1**,** l**,** r**,** val**);** **}** // Use this  T query**(**int at**,** int l**,** int r**,** int L**,** int R**)**  **{**  push**(**at**,** l**,** r**);**  **if** **(**L **<=** l **&&** r **<=** R**)** **return** tree**[**at**];**  int m **=** **(**l **+** r**)** **>>** 1**;**  **if** **(**R **<=** m**)** **return** query**(**at **<<** 1**,** l**,** m**,** L**,** R**);**  **if** **(**m **<** L**)** **return** query**(**at **<<** 1 **|** 1**,** m **+** 1**,** r**,** L**,** R**);**    **return** combine**(**query**(**at **<<** 1**,** l**,** m**,** L**,**  R**),** query**(**at **<<** 1 **|** 1**,** m **+** 1**,** r**,** L**,** R**));**  **}**  T Query**(**int l**,** int r**)** **{**  **return** query**(**1**,** 0**,** n **-** 1**,** l**,** r**);** **}**  **};**  void solve**(**ll cs**){**  ll j**,** i**,** p**,** q**,** a**,** b**,** c**,** m**,** n**,** k**;**  cin **>>** n**;**  vector**<**ll**>** v**(**n**);**  **for(**auto **&**x **:** v**)** cin **>>** x**;**  vector**<**seg\_node**>** vv**(**n**);**  segtree**<**seg\_node**>** tr **=** segtree**<**seg\_node**>(**n**);**  tr**.**Build**(**vv**);**  cin **>>** q**;**  **for(**i**=**0**;**i**<**q**;**i**++){**  ll ty**;**  cin **>>** ty **>>** a **>>** b**;**  **if(**ty**==**0**){**  tr**.**Update**(**a**,** b**,** node**);**  **}**  **else{**  auto r **=** tr**.**Query**(**a**,** b**);**  **}**  **}**  cout **<<** endl**;**  **}** |
| **//LCS**  int main**(){**  ll i**,** j**,** n**,** m**;**  string s1**,** s2**;**  cin **>>** s1 **>>** s2**;**  n **=** s1**.**size**();**  m **=** s2**.**size**();**  vector**<**vector**<**ll**>>** mem**(**n**+**1**,** vector**<**ll**>** **(**m**+**1**,** 0**));**  **for(**i**=**n**-**1**;**i**>-**1**;**i**--){**  **for(**j**=**m**-**1**;**j**>-**1**;**j**--){**  **if(**s1**[**i**]==**s2**[**j**]){**  mem**[**i**][**j**]** **=** 1 **+** mem**[**i**+**1**][**j**+**1**];**  **}**  **else{**  mem**[**i**][**j**]** **=** max**(**mem**[**i**+**1**][**j**],** mem**[**i**][**j**+**1**]);**  **}**  **}**  **}**  cout **<<** mem**[**0**][**0**]** **<<** endl**;**  **}** | **//Knapsack**  vector**<**ll**>** pv**,** wv**;**  ll mem**[**1000**][**1000**];**  ll dpop**(**ll i**,** ll n**,** ll w**){**  **if(**i**==**n**)** **return** 0**;**  **if(**w**==**0**)** **return** 0**;**  **if(**mem**[**i**][**w**]!=**0**)** **return** mem**[**i**][**w**];**  ll r1**=**0**,** r2**=**0**;**  **if(**w**-**wv**[**i**]>=**0**)** r1 **=** pv**[**i**]** **+** dpop**(**i**+**1**,** n**,** w**-**wv**[**i**]);**  r2 **=** dpop**(**i**+**1**,** n**,** w**);**  **return** mem**[**i**][**w**]** **=** max**(**r1**,** r2**);**  **}**  int main**(){**  ll i**,** num\_of\_int**,** j**,** k**,** weight**,** ans**;**  cin **>>** num\_of\_int **>>** weight**;**  pv **=** vector**<**ll**>(**num\_of\_int**);**  wv **=** vector**<**ll**>(**num\_of\_int**);**  **for(**auto **&**x **:** pv**)** cin **>>** x**;**  **for(**auto **&**x **:** wv**)** cin **>>** x**;**  ans **=** dpop**(**0**,** num\_of\_int**,** weight**);**  cout **<<** ans **<<** endl**;**  **}** |
| vector**<**ll**>** v**;**  ll mem**[**1000**][**1000**];**  **//COIN CHANGE**  ll dp**(**ll i**,** ll n**,** ll k**){**  **if(**i**==**n **and** k**!=**0**)** **return** 1e9**;**  **if(**k**==**0**)** **return** 0**;**  **if(**mem**[**i**][**k**]!=**0**)** **return** mem**[**i**][**k**];**  ll r1 **=** 1e9**,** r2 **=** 1e9**;**  **if(**k**-**v**[**i**]>=**0**)** r1 **=** 1**+**dp**(**i**+**1**,** n**,** k**-**v**[**i**]);**  r2 **=** dp**(**i**+**1**,** n**,** k**);**  **return** mem**[**i**][**k**]** **=** min**(**r1**,** r2**);**  **}** | vector**<**ll**>** v**;**  ll mem**[**1000**][**1000**];**  **//COIN CHANGE**  ll dpop**(**ll i**,** ll n**,** ll w**,** ll k**){**  **if(**w**<**0**)** **return** 1e9**;**  **if(**i**==**n **and** w**!=**0**)** **return** 1e9**;**  **if(**w**==**0**)** **return** 0**;**  **if(**mem**[**i**][**w**]!=**0**)** **return** mem**[**i**][**w**];**  ll ans **=** 1e9**;**  **for(**ll j**=**0**;**j**<=**k**;**j**++){**  ans **=** min**(**ans**,**  j**+**dpop**(**i**+**1**,** n**,** w**-**j**\***v**[**i**],** k**));**  **}**  **return** mem**[**i**][**w**]** **=** ans**;**  **}** |
| vector**<**ll**>** v**;**  ll mem**[**10000**];**  **//COIN CHANGE**  ll dpop**(**ll n**,** ll k**){**  **if(**k**<**0**)** **return** 1e9**;**  **if(**k**==**0**)** **return** 0**;**  **if(**mem**[**k**]!=**0**)** **return** mem**[**k**];**  ll ans **=** 1e9**;**  **for(**ll i**=**0**;**i**<**n**;**i**++){**  ans **=** min**(**ans**,** 1**+**dpop**(**n**,** k**-**v**[**i**]));**  **}**  **return** mem**[**k**]** **=** ans**;**  **}** | vector**<**vector**<**ll**>>** mem**(**1000**,**  vector**<**ll**>** **(**1000**,** **-**1**));**  **//STRING COMMON LCS Type**  int dp**(**int i**,** int j**,** string s1**,** string s2**){**  **if(**i**==**n**)** **return** m**-**j**;**  **if(**j**==**m**)** **return** n**-**i**;**  **if(**mem**[**i**][**j**]!=-**1**)** **return** mem**[**i**][**j**];**  int ans **=** 0**;**  **if(**s1**[**i**]==**s2**[**j**])**  ans **=** dp**(**i**+**1**,** j**+**1**,** s1**,** s2**);**  **else{**  ans **=** 1 **+** min**(**dp**(**i**+**1**,** j**,** s1**,** s2**),**  min**(**dp**(**i**,** j**+**1**,** s1**,** s2**),**  dp**(**i**+**1**,** j**+**1**,** s1**,** s2**)));**  **}**  **return** mem**[**i**][**j**]** **=** ans**;**  **}** |
| #define EMPTY\_VALUE -1  #define MAX\_N 10  #define INF 1061109567  **// BITMASK\_Subset\_DP**  **//Travelling\_Salesman**  int w**[**MAX\_N**][**MAX\_N**];**  int mem**[**MAX\_N**][**1**<<**MAX\_N**];**  int turnOn**(**int x**,** int pos**)** **{**  **return** N **|** **(**1**<<**pos**);**  **}**  bool isOn**(**int x **,**int pos**)** **{**  **return** **(**bool**)(**x **&** **(**1**<<**pos**));**  **}**  int n**;**  int f**(**int i**,** int mask**)** **{**  **if** **(**mask **==** **(**1**<<**n**)** **-** 1**)** **{**  **return** w**[**i**][**0**];**  **}** | **if** **(**mem**[**i**][**mask**]** **!=** **-**1**)** **{**  **return** mem**[**i**][**mask**];**  **}**    int ans **=** INF**;**  **for** **(**int j **=** 0**;**j **<** n**;**j**++)** **{**  **if** **(**w**[**i**][**j**]** **==** INF**)** **continue;**    **if** **(**isOn**(**mask**,**j**)** **==** 0**)** **{**  int result **=** f**(**j**,** turnOn**(**mask**,** j**))** **+** w**[**i**][**j**];**  ans **=** min**(**ans**,** result**);**  **}**  **}**    **return** mem**[**i**][**mask**]** **=** ans**;**  **}** |
| **// Knapsack (Tabulation)**  // Time Complexity O(n \* w)  // Space Complexity O(n \* w)  const int maxn **=** 102**;**  const int maxm **=** 1e5 **+** 5**;**  int n**,** w**;**  int dp**[**maxn**][**maxm**];**  int wt**[**maxn**],** val**[**maxn**];**  int32\_t main**()** **{**  ios\_base**::**sync\_with\_stdio**(**0**),** cin**.**tie**(**0**);**  cin **>>** n **>>** w**;**  **for** **(**int i **=** 1**;** i **<=** n**;** **++**i**)** **{**  cin **>>** wt**[**i**]** **>>** val**[**i**];**  **}**  **for** **(**int i **=** 1**;** i **<=** n**;** **++**i**)** **{**  **for** **(**int cap **=** 0**;** cap **<=** w**;** **++**cap**)** **{**  **if** **(**cap **<** wt**[**i**])** **{**  dp**[**i**][**cap**]** **=** dp**[**i **-** 1**][**cap**];**  **}**  **else** **{**  dp**[**i**][**cap**]** **=** max**(**val**[**i**]** **+** dp**[**i **-** 1**][**cap **-** wt**[**i**]],** dp**[**i **-** 1**][**cap**]);**  **}**  **}**  **}**  cout **<<** dp**[**n**][**w**]** **<<** '\n'**;**  **return** 0**;**  **}** | // **Subset Sum (Tabulation)**  // Time Complexity - O(n \* target)  const int maxn **=** 1e2 **+** 5**;**  const int maxm **=** 1e5 **+** 5**;**  int nums**[**maxn**];**  int dp**[**maxn**][**maxm**];**  int32\_t main**()** **{**  ios\_base**::**sync\_with\_stdio**(**0**),** cin**.**tie**(**0**);**  int n**,** m**;**  cin **>>** n **>>** m**;**  **for** **(**int i **=** 1**;** i **<=** n**;** **++**i**)** cin **>>** nums**[**i**];**  // 1. base case  dp**[**0**][**0**]** **=** 1**;**  **for** **(**int i **=** 1**;** i **<=** m**;** **++**i**)** dp**[**0**][**i**]** **=** 0**;**  **for** **(**int i **=** 1**;** i **<=** n**;** **++**i**)** dp**[**i**][**0**]** **=** 1**;**  **for** **(**int i **=** 1**;** i **<=** n**;** **++**i**)** **{**  **for** **(**int target **=** 1**;** target **<=** m**;** **++**target**)** **{**  int ans1 **=** dp**[**i **-** 1**][**target**];**  **if** **(**target **<** nums**[**i**])** **{**  dp**[**i**][**target**]** **=** ans1**;**  **}**  **else** **{**  int ans2 **=** dp**[**i **-** 1**][**target **-** nums**[**i**]];**  dp**[**i**][**target**]** **=** ans1 **||** ans2**;**  **}**  **}**  **}**  cout **<<** dp**[**n**][**m**]** **<<** '\n'**;**  **return** 0**;**  **}** |
| int lisNlogN**(**vector**<**int**>** const**&** a**)** **{**  int n **=** a**.**size**();**  const int INF **=** 1e9**;**  vector**<**int**>** d**(**n**+**1**,** INF**);**  d**[**0**]** **=** **-**INF**;**  **for** **(**int i **=** 0**;** i **<** n**;** i**++)** **{**  int l **=** upper\_bound**(**d**.**begin**(),**  d**.**end**(),** a**[**i**])** **-** d**.**begin**();**  **if** **(**d**[**l**-**1**]** **<** a**[**i**]** **&&** a**[**i**]** **<** d**[**l**])**  d**[**l**]** **=** a**[**i**];**  **}**  int ans **=** 0**;**  **for** **(**int l **=** 0**;** l **<=** n**;** l**++)** **{**  **if** **(**d**[**l**]** **<** INF**)**  ans **=** l**;**  **}**  **return** ans**;**  **}** | int LISNlogN**(){**  ll i**,** n**,** j**,** k**,** ans**;**  cin **>>** n**;**  vector**<**ll**>** v**(**n**);**  **for(**auto **&**x **:** v**)** cin **>>** x**;**  vector**<**ll**>** mem**;**  **for(**auto x **:** v**){**  ll ind **=** lower\_bound**(**mem**.**begin**(),** mem**.**end**(),** x**)** **-** mem**.**begin**();**  **if(**ind**==**mem**.**size**())** mem**.**push\_back**(**x**);**  **else** mem**[**ind**]** **=** x**;**  **for(**auto xx **:** mem**)** cout **<<** xx **<<** " "**;**  cout **<<** endl**;**  **}**  cout **<<** mem**.**size**();**  **}** |
| **// DIGIT DP**  pair**<**string**,** ll**>** dp**[**19**][**2**][**2**][**2**];**  pair**<**string**,** ll**>** DP**(**ll i**,** ll l**,** ll h**,** ll st**,** string **&**s1**,** string **&**s2**){**  **if(**i**==**s1**.**size**())** **return** **{**""**,** 1**};**  **if(**dp**[**i**][**l**][**h**][**st**].**se**!=-**1**)**  **return** dp**[**i**][**l**][**h**][**st**];**  ll s **=** s1**[**i**]-**'0'**,** e **=** s2**[**i**]-**'0'**;**  **if(**l**)** s **=** 0**;**  **if(**h**)** e **=** 9**;**  pair**<**string**,** ll**>** ans **=** **{**""**,** **-**1**};**  **for(**ll j**=**s**;**j**<=**e**;**j**++){**  ll is\_l **=** l**;**  **if(**j**>**s1**[**i**]-**'0'**)** is\_l **=** 1**;**  ll is\_h **=** h**;**  **if(**j**<**s2**[**i**]-**'0'**)** is\_h **=** 1**;**  ll is\_st **=** st**;**  **if(**j**!=**0**)** is\_st **=** 1**;**  pair**<**string**,** ll**>** pi **=** DP**(**i**+**1**,** is\_l**,** is\_h**,** is\_st**,** s1**,** s2**);**  char ch **=** '0'**+**j**;**  pi**.**fi **=** ch **+** pi**.**fi**;**  **if(**is\_st**)** pi**.**se **=** pi**.**se **\*** j**;**  //cout << pi.fi << " " << pi.se << endl;  **if(**ans**.**se**<**pi**.**se**)** ans **=** pi**;**  //ans %= mod;  **}**  **return** dp**[**i**][**l**][**h**][**st**]** **=** ans**;**  **}**  void solve**(**ll cs**)**  **{**  ll n**,** m**,** i**,** a**,** b**,** c**,** d**,** j**,** k**,** ans**=**0**,** mn **=** 1e15**,** mx**,** cnt**,** q**;**  string s**,** s1**,** s2**;**  cin **>>** a **>>** b**;**  //vector<ll> v(n);  s1 **=** to\_string**(**a**);**  s2 **=** to\_string**(**b**);**  s **=** ""**;**  **for(**i**=**s1**.**size**();**i**<**19**;**i**++)** s **+=** '0'**;**  s1 **=** s **+** s1**;**  s **=** ""**;**  **for(**i**=**s2**.**size**();**i**<**19**;**i**++)** s **+=** '0'**;**  s2 **=** s **+** s2**;**  **for(**auto **&**x **:** dp**)**  **for(**auto **&**xx **:** x**)**  **for(**auto **&**xxx **:** xx**)**  **for(**auto **&**xxxx **:** xxx**)**  xxxx **=** **{**""**,** **-**1**};**  pair**<**string**,** ll**>** pi **=** DP**(**0**,** 0**,** 0**,** 0**,** s1**,** s2**);**  bool ok **=** 1**;**  **for(**i**=**0**;**i**<**pi**.**fi**.**size**();**i**++){**  **if(**pi**.**fi**[**i**]==**'0' **and** ok**)** **continue;**  **else** ok **=** 0**;**  cout **<<** pi**.**fi**[**i**];**  **}**  cout **<<** endl**;**  **}** | **// DIGIT DP**  ll dp**[**51**][**17**][**17**][**17**][**2**];**  ll DP**(**ll i**,** ll c1**,** ll c2**,** ll c3**,** ll u**,** string **&**s**){**  //cout << sz << " ";  ll mx **=** max**(**c1**,** max**(**c2**,** c3**));**  mx **=** 3**\***mx **-** **(**c1**+**c2**+**c3**);**  **if(**51**-**i**<**mx**)** **return** 0**;**  **if(**i**==**s**.**size**())** **{**  **if(**c1**==**c2 **and** c2**==**c3 **and** c1**>=**1**)** **return** 1**;**  **else** **return** 0**;**  **}**  //Optimization technique  **if(**dp**[**i**][**c1**][**c2**][**c3**][**u**]!=-**1 **and** u**)** **return** dp**[**i**][**c1**][**c2**][**c3**][**u**];**  ll ans **=** 0**;**  **for(**ll j**=**0**;**j**<=**9**;**j**++){**  ll num **=** **(**s**[**i**]-**'0'**);**  **if(!**u **and** j**>**num**)** **break;**  ll is\_u **=** u**;**  **if(**j**<**num**)** is\_u **=** 1**;**  /\*  ll is\_st = st;  if(j!=0) is\_st = 1;  \*/  ll nc1 **=** c1 **+** **((**j**==**3**)** **?** 1 **:** 0**);**  ll nc2 **=** c2 **+** **((**j**==**6**)** **?** 1 **:** 0**);**  ll nc3 **=** c3 **+** **((**j**==**9**)** **?** 1 **:** 0**);**  ans **+=** DP**(**i**+**1**,** nc1**,** nc2**,** nc3**,** is\_u**,** s**);**  ans **%=** mod**;**  //cout << ans << " " << i << " " << pos << " " << j << endl;  **}**  **return** dp**[**i**][**c1**][**c2**][**c3**][**u**]** **=** ans**;**  **}**  void solve**(**ll cs**)**  **{**  ll n**,** m**,** i**,** a**,** b**,** c**,** d**,** j**,** k**,** ans**=**0**,** mn **=** 1e15**,** mx**,** cnt**,** q**;**  string s**,** s1**,** s2**;**  cin **>>** s1 **>>** s2**;**  mn **=** 51 **-** s1**.**size**(),** s **=** ""**;**  **while(**mn**--)** s **+=** '0'**;**  s1 **=** s **+** s1**;**  mn **=** 51 **-** s2**.**size**(),** s **=** ""**;**  **while(**mn**--)** s **+=** '0'**;**  s2 **=** s **+** s2**;**  ans **=** DP**(**0**,** 0**,** 0**,** 0**,** 0**,** s2**);**  ans **-=** DP**(**0**,** 0**,** 0**,** 0**,** 0**,** s1**);**  ll c1 **=** 0**,** c2 **=** 0**,** c3 **=** 0**;**  **for(**auto x **:** s1**)** **{**  c1 **+=** **((**x**==**'3'**)** **?** 1 **:** 0**);**  c2 **+=** **((**x**==**'6'**)** **?** 1 **:** 0**);**  c3 **+=** **((**x**==**'9'**)** **?** 1 **:** 0**);**  **}**  **if(**c1**==**c2 **and** c2**==**c3 **and** c1**>**0**)** ans**++;**  **if(**ans**<**0**)** ans **+=** mod**;**  cout **<<** ans**;**  cout **<<** endl**;**  **}** |
| **// KMP**  // Time Complexity - O(m + n)  vector**<**int**>** prefix\_function**(**string s**)** **{**  int n **=** **(**int**)**s**.**size**();**  vector**<**int**>** pi**(**n**,** 0**);**  **for** **(**int i **=** 1**;** i **<** n**;** **++**i**)** **{**  int j **=** pi**[**i **-** 1**];**  **while** **(**j **>** 0 **&&** s**[**i**]** **!=** s**[**j**])** **{**  j **=** pi**[**j **-** 1**];**  **}**  **if** **(**s**[**i**]** **==** s**[**j**])** **{**  **++**j**;**  **}**  pi**[**i**]** **=** j**;**  **}**  **return** pi**;**  **}**  int main**(**int argc**,** char const **\***argv**[])** **{**  ios\_base**::**sync\_with\_stdio**(false),** cin**.**tie**(nullptr);**    string s **=** "na"**;**  vector**<**int**>** prefix **=** prefix\_function**(**s**);**  string t **=** "apnacollege"**;**  int pos **=** **-**1**;**  int i **=** 0**,** j **=** 0**;**  **while** **(**i **<** **(**int**)**t**.**size**())** **{**  **if** **(**t**[**i**]** **==** s**[**j**])** **{**  **++**j**;**  **++**i**;**  **}**  **else** **{**  **if** **(**j **!=** 0**)** **{**  j **=** prefix**[**j **-** 1**];**  **}**  **else** **{**  **++**i**;**  **}**  **}**  **if** **(**j **==** **(**int**)**s**.**size**())** **{**  pos **=** i **-** **(**int**)**s**.**size**();**  **break;**  **}**  **}**  cout **<<** pos **<<** '\n'**;**  **return** 0**;**  **}** | **// Implementation of Rabin Carp String Matching Algorithm**  // https://github.com/Shafaet/Programming-Contest-Algorithms/blob/master/Useful%20C%2B%2B%20Libraries/rabin-carp.cpp  // Returns the index of the first match  // Complexity O(n+m), this is unsafe because it doesn't check for collisons  long long Hash **(**const string **&**s**,** int m**,** long long B**,** long long M**){**  long long h **=** 0 **,** power **=** 1**;**  **for(**int i **=** m **-** 1**;** i **>=** 0**;** i**--){**  h **=** h **+** **(**s**[**i**]** **\*** power**)** **%** M**;**  h **=** h **%** M**;**  power **=** **(**power **\*** B**)%**M**;**  **}**  **return** h**;**  **}**  int match **(**const string **&**text**,** const string **&**pattern**)** **{**  int n **=** text**.**size**();**  int m **=** pattern**.**size**();**  **if** **(**n **<** m**)** **return** **-**1**;**  **if** **(**m **==** 0 **or** n **==** 0**)** **{**  **return** **-**1**;**  **}**  long long B **=** 347**,** M **=** 1000000000 **+** 7**;**  // Calculate B^(m-1)  long long power **=** 1**;**  **for** **(**int i **=** 1**;** i **<=** m **-** 1**;** i**++)** **{**  power **=** **(**power **\*** B**)** **%** M**;**  **}**  // Find hash value of first m characters of text  // Find hash value of pattern  long long hash\_text **=** Hash**(**text**,** m**,** B**,** M**);**  long long hash\_pattern **=** Hash**(**pattern**,** m**,** B**,** M**);**  **if** **(**hash\_text **==** hash\_pattern**)** **{**  // returns the index of the match  **return** 0**;**  // We should've checked the substrings character by character here as hash collision might happen  **}**  **for** **(**int i **=** m**;** i **<** n**;** i**++){**  // Update Rolling Hash  hash\_text **=** **(**hash\_text **-** **(**power **\*** text**[**i **-** m**])** **%** M**)** **%** M**;**  hash\_text **=** **(**hash\_text **+** M**)** **%** M**;**  // take care of M of negative value  hash\_text **=** **(**hash\_text **\*** B**)** **%** M**;**  hash\_text **=** **(**hash\_text **+** text**[**i**])** **%** M**;**  **if** **(**hash\_text **==** hash\_pattern**)** **{**  **return** i **-** m **+** 1**;**  // returns the index of the match  // We should've checked the substrings character by character here as hash collision might happen  **}**  **}**  **return** **-**1**;**  **}**  int main**()** **{**  cout **<<** match**(**"HelloWorld"**,** "ello"**)** **<<** '\n'**;**  **return** 0**;**  **}** |
| **// HASHING**  const int N **=** 1e6 **+** 9**;**  const int MOD1 **=** 127657753**,** MOD2 **=** 987654319**;**  const int p1 **=** 137**,** p2 **=** 277**;**  int ip1**,** ip2**;**  pair**<**int**,** int**>** pw**[**N**],** ipw**[**N**];**  long long binpow**(**long long a**,** long long b**,** long long m**)** **{**  a **%=** m**;**  long long res **=** 1**;**  **while** **(**b **>** 0**)** **{**  **if** **(**b**&**1**)** **{**  res **=** res **\*** a **%** m**;**  **}**  a **=** a **\*** a **%** m**;**  b **>>=** 1**;**  **}**  **return** res**;**  **}**  void prec**()** **{**  pw**[**0**]** **=** **{**1**,** 1**};**  **for** **(**int i **=** 1**;** i **<** N**;** i**++)** **{**  pw**[**i**].**first **=** 1LL **\*** pw**[**i **-** 1**].**first **\*** p1 **%** MOD1**;**  pw**[**i**].**second **=** 1LL **\*** pw**[**i **-** 1**].**second **\*** p2 **%** MOD2**;**  **}**  ip1 **=** binpow**(**p1**,** MOD1 **-** 2**,** MOD1**);**  ip2 **=** binpow**(**p2**,** MOD2 **-** 2**,** MOD2**);**  ipw**[**0**]** **=** **{**1**,** 1**};**  **for** **(**int i **=** 1**;** i **<** N**;** i**++)** **{**  ipw**[**i**].**first **=** 1LL **\*** ipw**[**i **-** 1**].**first **\*** ip1 **%** MOD1**;**  ipw**[**i**].**second **=** 1LL **\*** ipw**[**i **-** 1**].**second **\***  ip2 **%** MOD2**;**  **}**  **}**  struct Hashing **{**  int n**;**  string s**;** // 0 - indexed  vector**<**pair**<**int**,** int**>>** hs**;** // 1 - indexed  Hashing**()** **{}**  Hashing**(**string \_s**)** **{**  n **=** \_s**.**size**();**  s **=** \_s**;**  hs**.**emplace\_back**(**0**,** 0**);**  **for** **(**int i **=** 0**;** i **<** n**;** i**++)** **{**  pair**<**int**,** int**>** p**;**  p**.**first **=** **(**hs**[**i**].**first **+** 1LL  **\*** pw**[**i**].**first **\*** s**[**i**]** **%** MOD1**)** **%** MOD1**;**  p**.**second **=** **(**hs**[**i**].**second **+** 1LL  **\*** pw**[**i**].**second **\*** s**[**i**]** **%** MOD2**)** **%** MOD2**;**  hs**.**push\_back**(**p**);**  **}**  **}**  pair**<**int**,** int**>** get\_hash**(**int l**,** int r**)** **{** // 1 - indexed  assert**(**1 **<=** l **&&** l **<=** r **&&** r **<=** n**);**  pair**<**int**,** int**>** ans**;**  ans**.**first **=** **(**hs**[**r**].**first **-** hs**[**l **-** 1**].**first **+** MOD1**)**  **\*** 1LL **\*** ipw**[**l **-** 1**].**first **%** MOD1**;**  ans**.**second **=** **(**hs**[**r**].**second **-** hs**[**l **-** 1**].**second **+** MOD2**)**  **\*** 1LL **\*** ipw**[**l **-** 1**].**second **%** MOD2**;**  **return** ans**;**  **}**  pair**<**int**,** int**>** get\_hash**()** **{**  **return** get\_hash**(**1**,** n**);**  **}**  **};**  int main**()** **{**  ios\_base**::**sync\_with\_stdio**(**0**),** cin**.**tie**(**0**);**  prec**();**  int n**;**  **while** **(**cin **>>** n**)** **{**  string s**,** p**;**  cin **>>** p **>>** s**;**  Hashing h**(**s**);**  auto hs **=** Hashing**(**p**).**get\_hash**();**  **for(**int i **=** 1**;** i **+** n **-** 1 **<=** **(**int**)**s**.**size**();** i**++)** **{**  **if** **(**h**.**get\_hash**(**i**,** i **+** n **-** 1**)** **==** hs**)**  cout **<<** i **-** 1 **<<** '\n'**;**  **}**  cout **<<** '\n'**;**  **}**  **return** 0**;**  **}** |  |
| **// MISC. Algo**  **// MinMaxSubSum**  void solve**(**ll cs**){**  ll j**,** i**,** p**,** q**,** a**,** b**,** c**,** d**,** m**,** n**,** k**,** mn **=** 0**,** mx **=** 998244353**;**  string s**,** s1**,** s2**;**  cin **>>** n**;**  vector**<**ll**>** v**(**n**),** pre**(**n**);**  **for(**auto **&**x **:** v**)** cin **>>** x**;**  // at least length 2  mn **=** 0**,** mx **=** 0**;**  ll sum **=** 0**;**  **for(**i**=**0**;**i**<**n**;**i**++){**  sum **+=** v**[**i**];**  pre**[**i**]** **=** sum**;**  **if(-**1**<**i**-**2**)** mx **=** max**(**mx**,** pre**[**i**-**2**]);**  **if(**i**>**0**)** mn **=** min**(**mn**,** sum**-**mx**);**  **}**  p **=** 0**;**  **for(**auto x **:** v**)** p **+=** x**;**  p **-=** mn**;**  cout **<<** p**+**abs**(**mn**)** **;**  cout **<<** endl**;**  **}** |  |