Contents

[**1.BigNum** 1](#_Toc121094373)

[**2.Math** 2](#_Toc121094374)

[ *Phi hàm euler* 2](#_Toc121094375)

[ *Lucas Theorem* 2](#_Toc121094376)

[ *Bao hàm – Loại trừ* 3](#_Toc121094377)

[ *Xác suất* 3](#_Toc121094378)

[ *Guass* 4](#_Toc121094379)

[ *FFT* 4](#_Toc121094380)

[ *NTT* 5](#_Toc121094381)

[ *CRT* 6](#_Toc121094382)

[ *Pollard Rho* 6](#_Toc121094383)

[ *Bitmask* 7](#_Toc121094384)

[ *Catalan* 8](#_Toc121094385)

[ *Phi hàm euler* 8](#_Toc121094386)

[ *Extend euclid* 8](#_Toc121094387)

[ *Công thức tính sum* 8](#_Toc121094388)

[ *Tính tổng các số nguyên tố nhỏ hơn hoặc bằng n* 8](#_Toc121094389)

[**3. Hình học** 8](#_Toc121094390)

[ *Hình tròn* 8](#_Toc121094391)

[ *Đường thẳng* 9](#_Toc121094392)

[ *Tam giác* 10](#_Toc121094393)

[ *Bao lồi* 10](#_Toc121094394)

[ *Diện tích đa giác* 10](#_Toc121094395)

[ *Kiểm tra 1 điểm nằm trong đa giác* 10](#_Toc121094396)

[**4.Data Structures** 10](#_Toc121094397)

[ *Segment Tree* 10](#_Toc121094398)

[ *Trie* 11](#_Toc121094399)

[ *Ordered* 11](#_Toc121094400)

[ *Segment Tree (ver2)* 12](#_Toc121094401)

[ *SegmentTree ver3* 12](#_Toc121094402)

[ *BIT 2D* 12](#_Toc121094403)

[ *MO* 13](#_Toc121094404)

[ *Squareroot decomposition* 13](#_Toc121094405)

[**5. Graph** 13](#_Toc121094406)

[ *2sat* 13](#_Toc121094407)

[ *Check đồ thị hai phía* 14](#_Toc121094408)

[ *Topo sort* 14](#_Toc121094409)

[ *DSU* 15](#_Toc121094410)

[ *LCA* 15](#_Toc121094411)

[ *Euler Tour* 15](#_Toc121094412)

[ *Heavy Light* 16](#_Toc121094413)

[ *Tarjan – TPLT mạnh* 16](#_Toc121094414)

[ *Khớp, cầu* 17](#_Toc121094415)

[ *Dijkstra on Segment Tree* 17](#_Toc121094416)

[ *Cạnh min, max giữa hai đỉnh x, y bất kì trên cây* 18](#_Toc121094417)

[ *Dinic* 18](#_Toc121094418)

[ *Fenwick Tree on Tree* 19](#_Toc121094419)

[ *DSU on Tree* 20](#_Toc121094420)

[**6. String** 20](#_Toc121094421)

[ *Hashing use with sum prefix* 20](#_Toc121094422)

[ *Hasing use with segment tree* 21](#_Toc121094423)

[ *KMP* 21](#_Toc121094424)

# **1.BigNum**

|  |
| --- |
| string s1**,**s2**;**  string bigsum**(**string num1**,** string num2**)**  **{**  string s **=** ""**;**  int len1 **=** num1**.**length**(),** len2 **=** num2**.**length**();**  reverse**(**num1**.**begin**(),** num1**.**end**());**  reverse**(**num2**.**begin**(),** num2**.**end**());**  **if(**len1 **>** len2**)**  **{**  swap**(**num1**,** num2**);**  swap**(**len1**,** len2**);**  **}**  **while(**num1**.**length**()** **<** num2**.**length**())**  **{**  num1 **+=** '0'**;**  **}**  int mem **=** 0**;**  **for(**int i **=** 0**;** i **<** len2**;** i**++)**  **{**  int sum **=** **(**num1**[**i**]** **-** 48**)** **+** **(**num2**[**i**]** **-** 48**)** **+** mem**;**  s**.**push\_back**(**sum **%** 10 **+** '0'**);**  mem **=** sum **/** 10**;**  **}**  **if(**mem**)** s**.**push\_back**(**mem **+** '0'**);**  reverse**(**s**.**begin**(),** s**.**end**()** **);**  **return** s**;**  **}**  string multiply**(**string num1**,** string num2**)**  **{**  int len1 **=** num1**.**size**();**  int len2 **=** num2**.**size**();**  **if** **(**len1 **==** 0 **||** len2 **==** 0**)**  **return** "0"**;**  vector**<**int**>** result**(**len1 **+** len2**,** 0**);**  int i\_n1 **=** 0**;**  int i\_n2 **=** 0**;**  **for** **(**int i**=**len1**-**1**;** i**>=**0**;** i**--)**  **{**  int carry **=** 0**;**  int n1 **=** num1**[**i**]** **-** '0'**;**  i\_n2 **=** 0**;**  **for** **(**int j**=**len2**-**1**;** j**>=**0**;** j**--)**  **{**  int n2 **=** num2**[**j**]** **-** '0'**;**  int sum **=** n1**\***n2 **+** result**[**i\_n1 **+** i\_n2**]** **+** carry**;**  carry **=** sum**/**10**;**  result**[**i\_n1 **+** i\_n2**]** **=** sum **%** 10**;**  i\_n2**++;**  **}**  **if** **(**carry **>** 0**)**  result**[**i\_n1 **+** i\_n2**]** **+=** carry**;**  i\_n1**++;**  **}**  int i **=** result**.**size**()** **-** 1**;**  **while** **(**i**>=**0 **&&** result**[**i**]** **==** 0**)**  i**--;**  **if** **(**i **==** **-**1**)**  **return** "0"**;**  string s **=** ""**;**    **while** **(**i **>=** 0**)**  s **+=** to\_string**(**result**[**i**--]);**    **return** s**;**  **}**  long long modBigNumber**(**string num**,** long long m**)**  **{**  vector**<**int**>** vec**;**  long long mod **=** 0**;**  **for** **(**int i **=** 0**;** i **<** num**.**size**();** i**++)** **{**    int digit **=** num**[**i**]** **-** '0'**;**    mod **=** mod **\*** 10 **+** digit**;**    int quo **=** mod **/** m**;**  vec**.**push\_back**(**quo**);**    mod **=** mod **%** m**;**  **}**  **return** mod**;**  bool zeroflag **=** 0**;**  **for** **(**int i **=** 0**;** i **<** vec**.**size**();** i**++)** **{**  **if** **(**vec**[**i**]** **==** 0 **&&** zeroflag **==** 0**)**  **continue;**  zeroflag **=** 1**;**  cout **<<** vec**[**i**];**  **}**  **}**  long long pow**(**string num**,** long long x**,** long long m**)**  **{**  **if(**x **==** 0**)** **return** 1 **%** m**;**  **if(**x **==** 1**)** **return** modBigNumber**(**num**,** m**);**  long long mid **=** pow**(**num**,** x **/** 2**,** m**)** **%** m**;**  mid **=** mid **\*** mid **%** m**;**  **if(**x **%** 2 **==** 0**)** **return** mid**;**  **return** mid **\*** modBigNumber**(**num**,** m**)** **%** m**;**  **}**  string bigsub**(**string num1**,** string num2**)**  **{**  string s **=** ""**;**  int len1 **=** num1**.**length**(),** len2 **=** num2**.**length**();**  **if(**len1 **<** len2 **||** len1 **==** len2 **&&** num1 **<** num2**)**  **{**  swap**(**num1**,** num2**);**  swap**(**len1**,** len2**);**  **}**  reverse**(**num1**.**begin**(),** num1**.**end**());**  reverse**(**num2**.**begin**(),** num2**.**end**());**    **while(**num1**.**length**()** **>** num2**.**length**())**  **{**  num2 **+=** '0'**;**  **}**  int mem **=** 0**;**  **for(**int i **=** 0**;** i **<** len1**;** i**++)**  **{**  int sum **=** **(**num1**[**i**]** **-** 48**)** **-** **(**num2**[**i**]** **-** 48**)** **-** mem**;**  **if(**sum **<** 0**)**  **{**  mem **=** 1**;**  sum **+=** 10**;**  **}**  **else** mem **=** 0**;**  s**.**push\_back**(**sum **+** '0'**);**  **}**  **for(**int i **=** s**.**length**()** **-** 1**;** i **>=**0**;** i**--)**  **{**  **if(**s**[**i**]** **==** '0'**)**  **{**  s**.**pop\_back**();**  **continue;**  **}**  **break;**  **}**  **if(**s**.**empty**())** s**.**push\_back**(**'0'**);**  reverse**(**s**.**begin**(),** s**.**end**()** **);**  **return** s**;**  **}** |

# **2.Math**

|  |  |
| --- | --- |
| |  | | --- | | * *Phi hàm euler* |   int eulerPhi**(**int n**)** **{** // = n (1-1/p1) ... (1-1/pn)  **if** **(**n **==** 0**)** **return** 0**;**  int ans **=** n**;**  **for** **(**int x **=** 2**;** x**\***x **<=** n**;** **++**x**)** **{**  **if** **(**n **%** x **==** 0**)** **{**  ans **-=** ans **/** x**;**  **while** **(**n **%** x **==** 0**)** n **/=** x**;**  **}**  **}**  **if** **(**n **>** 1**)** ans **-=** ans **/** n**;**  **return** ans**;**  **}** |
| |  | | --- | | * *Lucas Theorem* |   #include <bits/stdc++.h>  **using** **namespace** std**;**  const int N **=** 1e6 **+** 3**,** mod **=** 1e6 **+** 3**;**  **using** ll **=** long long**;**  template **<**const int32\_t MOD**>**  struct modint  **{**  int32\_t value**;**  modint**()** **=** **default;**  modint**(**int32\_t value\_**)** **:** value**(**value\_**)** **{}**  inline modint**<**MOD**>** **operator+(**modint**<**MOD**>** other**)** const  **{**  int32\_t c **=** **this->**value **+** other**.**value**;**  **return** modint**<**MOD**>(**c **>=** MOD **?** c **-** MOD **:** c**);**  **}**  inline modint**<**MOD**>** **operator-(**modint**<**MOD**>** other**)** const  **{**  int32\_t c **=** **this->**value **-** other**.**value**;**  **return** modint**<**MOD**>(**c **<** 0 **?** c **+** MOD **:** c**);**  **}**  inline modint**<**MOD**>** **operator\*(**modint**<**MOD**>** other**)** const  **{**  int32\_t c **=** **(**int64\_t**)this->**value **\*** other**.**value **%** MOD**;**  **return** modint**<**MOD**>(**c **<** 0 **?** c **+** MOD **:** c**);**  **}**  inline modint**<**MOD**>** **&operator+=(**modint**<**MOD**>** other**)**  **{**  **this->**value **+=** other**.**value**;**  **if** **(this->**value **>=** MOD**)**  **this->**value **-=** MOD**;**  **return** **\*this;**  **}**  inline modint**<**MOD**>** **&operator-=(**modint**<**MOD**>** other**)**  **{**  **this->**value **-=** other**.**value**;**  **if** **(this->**value **<** 0**)**  **this->**value **+=** MOD**;**  **return** **\*this;**  **}**  inline modint**<**MOD**>** **&operator\*=(**modint**<**MOD**>** other**)**  **{**  **this->**value **=** **(**int64\_t**)this->**value **\*** other**.**value **%** MOD**;**  **if** **(this->**value **<** 0**)**  **this->**value **+=** MOD**;**  **return** **\*this;**  **}**  inline modint**<**MOD**>** **operator-()** const **{** **return** modint**<**MOD**>(this->**value **?** MOD **-** **this->**value **:** 0**);** **}**  modint**<**MOD**>** pow**(**uint64\_t k**)** const  **{**  modint**<**MOD**>** x **=** **\*this,** y **=** 1**;**  **for** **(;** k**;** k **>>=** 1**)**  **{**  **if** **(**k **&** 1**)**  y **\*=** x**;**  x **\*=** x**;**  **}**  **return** y**;**  **}**  modint**<**MOD**>** inv**()** const **{** **return** pow**(**MOD **-** 2**);** **}** // MOD must be a prime  inline modint**<**MOD**>** **operator/(**modint**<**MOD**>** other**)** const **{** **return** **\*this** **\*** other**.**inv**();** **}**  inline modint**<**MOD**>** **operator/=(**modint**<**MOD**>** other**)** **{** **return** **\*this** **\*=** other**.**inv**();** **}**  inline bool **operator==(**modint**<**MOD**>** other**)** const **{** **return** value **==** other**.**value**;** **}**  inline bool **operator!=(**modint**<**MOD**>** other**)** const **{** **return** value **!=** other**.**value**;** **}**  inline bool **operator<(**modint**<**MOD**>** other**)** const **{** **return** value **<** other**.**value**;** **}**  inline bool **operator>(**modint**<**MOD**>** other**)** const **{** **return** value **>** other**.**value**;** **}**  **};**  template **<**int32\_t MOD**>**  modint**<**MOD**>** **operator\*(**int32\_t value**,** modint**<**MOD**>** n**)** **{** **return** modint**<**MOD**>(**value**)** **\*** n**;** **}**  template **<**int32\_t MOD**>**  modint**<**MOD**>** **operator\*(**int64\_t value**,** modint**<**MOD**>** n**)** **{** **return** modint**<**MOD**>(**value **%** MOD**)** **\*** n**;** **}**  template **<**int32\_t MOD**>**  istream **&operator>>(**istream **&**in**,** modint**<**MOD**>** **&**n**)** **{** **return** in **>>** n**.**value**;** **}**  template **<**int32\_t MOD**>**  ostream **&operator<<(**ostream **&**out**,** modint**<**MOD**>** n**)** **{** **return** out **<<** n**.**value**;** **}**  **using** mint **=** modint**<**mod**>;**  struct combi  **{**  int n**;**  vector**<**mint**>** facts**,** finvs**,** invs**;**  combi**(**int \_n**)** **:** n**(**\_n**),** facts**(**\_n**),** finvs**(**\_n**),** invs**(**\_n**)**  **{**  facts**[**0**]** **=** finvs**[**0**]** **=** 1**;**  invs**[**1**]** **=** 1**;**  **for** **(**int i **=** 2**;** i **<** n**;** i**++)**  invs**[**i**]** **=** invs**[**mod **%** i**]** **\*** **(-**mod **/** i**);**  **for** **(**int i **=** 1**;** i **<** n**;** i**++)**  **{**  facts**[**i**]** **=** facts**[**i **-** 1**]** **\*** i**;**  finvs**[**i**]** **=** finvs**[**i **-** 1**]** **\*** invs**[**i**];**  **}**  **}**  inline mint fact**(**int n**)** **{** **return** facts**[**n**];** **}**  inline mint finv**(**int n**)** **{** **return** finvs**[**n**];** **}**  inline mint inv**(**int n**)** **{** **return** invs**[**n**];** **}**  inline mint ncr**(**int n**,** int k**)** **{** **return** n **<** k **or** k **<** 0 **?** 0 **:** facts**[**n**]** **\*** finvs**[**k**]** **\*** finvs**[**n **-** k**];** **}**  **};**  combi C**(**N**);**  // returns nCr modulo mod where mod is a prime  // Complexity: log(n)  mint lucas**(**ll n**,** ll r**)**  **{**  **if** **(**r **>** n**)**  **return** 0**;**  **if** **(**n **<** mod**)**  **return** C**.**ncr**(**n**,** r**);**  **return** lucas**(**n **/** mod**,** r **/** mod**)** **\*** lucas**(**n **%** mod**,** r **%** mod**);**  **}**  int32\_t main**()**  **{**  ios\_base**::**sync\_with\_stdio**(**0**);**  cin**.**tie**(**0**);**  cout **<<** lucas**(**100000000**,** 2322**)** **<<** '\n'**;**  **return** 0**;**  **}** |
| |  | | --- | | * *Bao hàm – Loại trừ* |   int cal**(**int n**,** int r**)**  **{**  int sum **=** 0**;**  vector**<**int**>** p**;**  **for** **(**int i **=** 2**;** i **\*** i **<=** n**;** **++**i**)**  **if** **(**n **%** i **==** 0**)**  **{**  p**.**push\_back**(**i**);**  **while** **(**n **%** i **==** 0**)**  n **/=** i**;**  **}**  **if** **(**n **>** 1**)**  p**.**push\_back**(**n**);**  **for** **(**int msk **=** 1**;** msk **<** **(**1 **<<** p**.**size**());** **++**msk**)**  **{**  int mult **=** 1**,** bits **=** 0**;**  **for** **(**int i **=** 0**;** i **<** **(**int**)**p**.**size**();** **++**i**)**  **if** **(**msk **&** **(**1 **<<** i**))**  **{**  **++**bits**;**  mult **\*=** p**[**i**];**  **}**  int cur **=** r **/** mult**;**  **if** **(**bits **%** 2 **==** 1**)**  sum **+=** cur**;**  **else**  sum **-=** cur**;**  **}**  **return** r **-** sum**;**  **}** |
| |  | | --- | | * *Xác suất* |   Xác suất có điều kiện (Conditional Probability):    Quy tắc nhân:   * Nếu A và B là hai biến cố phụ thuộc: P(AB) = P(A∪B) − (PA + PB) * Nếu A và B là hai biến cố độc lập: P(AB) = P(A).P(B).   Quy tắc cộng: P(A∪B) = P(A) + P(B) − P(AB)  Bayes:    Bayes mở rộng:    Bernoulli: xác suất để biến cố A xảy ra đúng k lần trong n phép thử độc lập: |
| |  | | --- | | * *Guass* |   double EPS **=** 0.000001**;**  int INF **=** INFINITY**;**  int gauss **(**vector **<** vector**<**double**>** **>** a**,** vector**<**double**>** **&** ans**)** **{**  int n **=** **(**int**)** a**.**size**();**  int m **=** **(**int**)** a**[**0**].**size**()** **-** 1**;**    vector**<**int**>** where **(**m**,** **-**1**);**  **for** **(**int col**=**0**,** row**=**0**;** col**<**m **&&** row**<**n**;** **++**col**)** **{**  int sel **=** row**;**  **for** **(**int i**=**row**;** i**<**n**;** **++**i**)**  **if** **(**abs **(**a**[**i**][**col**])** **>** abs **(**a**[**sel**][**col**]))**  sel **=** i**;**  **if** **(**abs **(**a**[**sel**][**col**])** **<** EPS**)**  **continue;**  **for** **(**int i**=**col**;** i**<=**m**;** **++**i**)**  swap **(**a**[**sel**][**i**],** a**[**row**][**i**]);**  where**[**col**]** **=** row**;**    **for** **(**int i**=**0**;** i**<**n**;** **++**i**)**  **if** **(**i **!=** row**)** **{**  double c **=** a**[**i**][**col**]** **/** a**[**row**][**col**];**  **for** **(**int j**=**col**;** j**<=**m**;** **++**j**)**  a**[**i**][**j**]** **-=** a**[**row**][**j**]** **\*** c**;**  **}**  **++**row**;**  **}**    ans**.**assign **(**m**,** 0**);**  **for** **(**int i**=**0**;** i**<**m**;** **++**i**)**  **if** **(**where**[**i**]** **!=** **-**1**)**  ans**[**i**]** **=** a**[**where**[**i**]][**m**]** **/** a**[**where**[**i**]][**i**];**  **for** **(**int i**=**0**;** i**<**n**;** **++**i**)** **{**  double sum **=** 0**;**  **for** **(**int j**=**0**;** j**<**m**;** **++**j**)**  sum **+=** ans**[**j**]** **\*** a**[**i**][**j**];**  **if** **(**abs **(**sum **-** a**[**i**][**m**])** **>** EPS**)**  **return** 0**;**  **}**    **for** **(**int i**=**0**;** i**<**m**;** **++**i**)**  **if** **(**where**[**i**]** **==** **-**1**)**  **return** INF**;**  **return** 1**;**  **}** |
| |  | | --- | | * *FFT* |   #include <bits/stdc++.h>  **using** **namespace** std**;**  const int N **=** 3e5 **+** 9**;**  const double PI **=** acos**(-**1**);**  struct base  **{**  double a**,** b**;**  base**(**double a **=** 0**,** double b **=** 0**)** **:** a**(**a**),** b**(**b**)** **{}**  const base **operator+(**const base **&**c**)** const  **{**  **return** base**(**a **+** c**.**a**,** b **+** c**.**b**);**  **}**  const base **operator-(**const base **&**c**)** const  **{**  **return** base**(**a **-** c**.**a**,** b **-** c**.**b**);**  **}**  const base **operator\*(**const base **&**c**)** const  **{**  **return** base**(**a **\*** c**.**a **-** b **\*** c**.**b**,** a **\*** c**.**b **+** b **\*** c**.**a**);**  **}**  **};**  void fft**(**vector**<**base**>** **&**p**,** bool inv **=** 0**)**  **{**  int n **=** p**.**size**(),** i **=** 0**;**  **for** **(**int j **=** 1**;** j **<** n **-** 1**;** **++**j**)**  **{**  **for** **(**int k **=** n **>>** 1**;** k **>** **(**i **^=** k**);** k **>>=** 1**)**  **;**  **if** **(**j **<** i**)**  swap**(**p**[**i**],** p**[**j**]);**  **}**  **for** **(**int l **=** 1**,** m**;** **(**m **=** l **<<** 1**)** **<=** n**;** l **<<=** 1**)**  **{**  double ang **=** 2 **\*** PI **/** m**;**  base wn **=** base**(**cos**(**ang**),** **(**inv **?** 1. **:** **-**1.**)** **\*** sin**(**ang**)),** w**;**  **for** **(**int i **=** 0**,** j**,** k**;** i **<** n**;** i **+=** m**)**  **{**  **for** **(**w **=** base**(**1**,** 0**),** j **=** i**,** k **=** i **+** l**;** j **<** k**;** **++**j**,** w **=** w **\*** wn**)**  **{**  base t **=** w **\*** p**[**j **+** l**];**  p**[**j **+** l**]** **=** p**[**j**]** **-** t**;**  p**[**j**]** **=** p**[**j**]** **+** t**;**  **}**  **}**  **}**  **if** **(**inv**)**  **for** **(**int i **=** 0**;** i **<** n**;** **++**i**)**  p**[**i**].**a **/=** n**,** p**[**i**].**b **/=** n**;**  **}**  vector**<**long long**>** multiply**(**vector**<**int**>** **&**a**,** vector**<**int**>** **&**b**)**  **{**  int n **=** a**.**size**(),** m **=** b**.**size**(),** t **=** n **+** m **-** 1**,** sz **=** 1**;**  **while** **(**sz **<** t**)**  sz **<<=** 1**;**  vector**<**base**>** x**(**sz**),** y**(**sz**),** z**(**sz**);**  **for** **(**int i **=** 0**;** i **<** sz**;** **++**i**)**  **{**  x**[**i**]** **=** i **<** **(**int**)**a**.**size**()** **?** base**(**a**[**i**],** 0**)** **:** base**(**0**,** 0**);**  y**[**i**]** **=** i **<** **(**int**)**b**.**size**()** **?** base**(**b**[**i**],** 0**)** **:** base**(**0**,** 0**);**  **}**  fft**(**x**),** fft**(**y**);**  **for** **(**int i **=** 0**;** i **<** sz**;** **++**i**)**  z**[**i**]** **=** x**[**i**]** **\*** y**[**i**];**  fft**(**z**,** 1**);**  vector**<**long long**>** ret**(**sz**);**  **for** **(**int i **=** 0**;** i **<** sz**;** **++**i**)**  ret**[**i**]** **=** **(**long long**)(**z**[**i**].**a **+** 0.5**);**  **while** **((**int**)**ret**.**size**()** **>** 1 **&&** ret**.**back**()** **==** 0**)**  ret**.**pop\_back**();**  **return** ret**;**  **}**  long long ans**[**N**];**  int32\_t main**()**  **{**  ios\_base**::**sync\_with\_stdio**(**0**);**  cin**.**tie**(**0**);**  int n**,** x**;**  cin **>>** n **>>** x**;**  vector**<**int**>** a**(**n **+** 1**,** 0**),** b**(**n **+** 1**,** 0**),** c**(**n **+** 1**,** 0**);**  int nw **=** 0**;**  a**[**0**]++;**  b**[**n**]++;**  long long z **=** 0**;**  **for** **(**int i **=** 1**;** i **<=** n**;** i**++)**  **{**  int k**;**  cin **>>** k**;**  nw **+=** k **<** x**;**  a**[**nw**]++;**  b**[-**nw **+** n**]++;**  z **+=** c**[**nw**]** **+** **!**nw**;**  c**[**nw**]++;**  **}**  auto res **=** multiply**(**a**,** b**);**  **for** **(**int i **=** n **+** 1**;** i **<** res**.**size**();** i**++)**  **{**  ans**[**i **-** n**]** **+=** res**[**i**];**  **}**  ans**[**0**]** **=** z**;**  **for** **(**int i **=** 0**;** i **<=** n**;** i**++)**  cout **<<** ans**[**i**]** **<<** ' '**;**  cout **<<** '\n'**;**  **return** 0**;**  **}**  // https://codeforces.com/contest/993/problem/E |
| |  | | --- | | * *NTT* |   #include <bits/stdc++.h>  **using** **namespace** std**;**  const int N **=** 1 **<<** 18**;**  const int mod **=** 998244353**;**  const int root **=** 3**;**  int lim**,** rev**[**N**],** w**[**N**],** wn**[**N**],** inv\_lim**;**  void reduce**(**int **&**x**)** **{** x **=** **(**x **+** mod**)** **%** mod**;** **}**  int POW**(**int x**,** int y**,** int ans **=** 1**)**  **{**  **for** **(;** y**;** y **>>=** 1**,** x **=** **(**long long**)**x **\*** x **%** mod**)**  **if** **(**y **&** 1**)**  ans **=** **(**long long**)**ans **\*** x **%** mod**;**  **return** ans**;**  **}**  void precompute**(**int len**)**  **{**  lim **=** wn**[**0**]** **=** 1**;**  int s **=** **-**1**;**  **while** **(**lim **<** len**)**  lim **<<=** 1**,** **++**s**;**  **for** **(**int i **=** 0**;** i **<** lim**;** **++**i**)**  rev**[**i**]** **=** rev**[**i **>>** 1**]** **>>** 1 **|** **(**i **&** 1**)** **<<** s**;**  const int g **=** POW**(**root**,** **(**mod **-** 1**)** **/** lim**);**  inv\_lim **=** POW**(**lim**,** mod **-** 2**);**  **for** **(**int i **=** 1**;** i **<** lim**;** **++**i**)**  wn**[**i**]** **=** **(**long long**)**wn**[**i **-** 1**]** **\*** g **%** mod**;**  **}**  void ntt**(**vector**<**int**>** **&**a**,** int typ**)**  **{**  **for** **(**int i **=** 0**;** i **<** lim**;** **++**i**)**  **if** **(**i **<** rev**[**i**])**  swap**(**a**[**i**],** a**[**rev**[**i**]]);**  **for** **(**int i **=** 1**;** i **<** lim**;** i **<<=** 1**)**  **{**  **for** **(**int j **=** 0**,** t **=** lim **/** i **/** 2**;** j **<** i**;** **++**j**)**  w**[**j**]** **=** wn**[**j **\*** t**];**  **for** **(**int j **=** 0**;** j **<** lim**;** j **+=** i **<<** 1**)**  **{**  **for** **(**int k **=** 0**;** k **<** i**;** **++**k**)**  **{**  const int x **=** a**[**k **+** j**],** y **=** **(**long long**)**a**[**k **+** j **+** i**]** **\*** w**[**k**]** **%** mod**;**  reduce**(**a**[**k **+** j**]** **+=** y **-** mod**),** reduce**(**a**[**k **+** j **+** i**]** **=** x **-** y**);**  **}**  **}**  **}**  **if** **(!**typ**)**  **{**  reverse**(**a**.**begin**()** **+** 1**,** a**.**begin**()** **+** lim**);**  **for** **(**int i **=** 0**;** i **<** lim**;** **++**i**)**  a**[**i**]** **=** **(**long long**)**a**[**i**]** **\*** inv\_lim **%** mod**;**  **}**  **}**  vector**<**int**>** multiply**(**vector**<**int**>** **&**f**,** vector**<**int**>** **&**g**)**  **{**  int n **=** **(**int**)**f**.**size**()** **+** **(**int**)**g**.**size**()** **-** 1**;**  precompute**(**n**);**  vector**<**int**>** a **=** f**,** b **=** g**;**  a**.**resize**(**lim**);**  b**.**resize**(**lim**);**  ntt**(**a**,** 1**),** ntt**(**b**,** 1**);**  **for** **(**int i **=** 0**;** i **<** lim**;** **++**i**)**  a**[**i**]** **=** **(**long long**)**a**[**i**]** **\*** b**[**i**]** **%** mod**;**  ntt**(**a**,** 0**);**  // while((int)a.size() && a.back() == 0) a.pop\_back();  **return** a**;**  **}**  int fact**[**N**],** ifact**[**N**];**  vector**<**int**>** shift**(**vector**<**int**>** **&**f**,** int c**)**  **{** // f(x + c)  int n **=** **(**int**)**f**.**size**();**  precompute**(**n **+** n **-** 1**);**  vector**<**int**>** a **=** f**;**  a**.**resize**(**lim**);**  **for** **(**int i **=** 0**;** i **<** n**;** **++**i**)**  a**[**i**]** **=** **(**long long**)**a**[**i**]** **\*** fact**[**i**]** **%** mod**;**  reverse**(**a**.**begin**(),** a**.**begin**()** **+** n**);**  vector**<**int**>** b**;**  b**.**resize**(**lim**);**  b**[**0**]** **=** 1**;**  **for** **(**int i **=** 1**;** i **<** n**;** **++**i**)**  b**[**i**]** **=** **(**long long**)**b**[**i **-** 1**]** **\*** c **%** mod**;**  **for** **(**int i **=** 0**;** i **<** n**;** **++**i**)**  b**[**i**]** **=** **(**long long**)**b**[**i**]** **\*** ifact**[**i**]** **%** mod**;**  ntt**(**a**,** 1**),** ntt**(**b**,** 1**);**  **for** **(**int i **=** 0**;** i **<** lim**;** **++**i**)**  a**[**i**]** **=** **(**long long**)**a**[**i**]** **\*** b**[**i**]** **%** mod**;**  ntt**(**a**,** 0**),** reverse**(**a**.**begin**(),** a**.**begin**()** **+** n**);**  vector**<**int**>** g**;**  g**.**resize**(**n**);**  **for** **(**int i **=** 0**;** i **<** n**;** **++**i**)**  g**[**i**]** **=** **(**long long**)**a**[**i**]** **\*** ifact**[**i**]** **%** mod**;**  **return** g**;**  **}**  vector**<**int**>** range\_mul**(**int n**)**  **{** //(x+1)\*(x+2)\*(x+3)...(x+n)  **if** **(**n **==** 0**)**  **return** vector**<**int**>({**1**});**  **if** **(**n **&** 1**)**  **{**  vector**<**int**>** f **=** range\_mul**(**n **-** 1**);**  f**.**push\_back**(**0**);**  **for** **(**int i **=** **(**int**)**f**.**size**()** **-** 1**;** i**;** **--**i**)**  f**[**i**]** **=** **(**f**[**i **-** 1**]** **+** **(**long long**)**n **\*** f**[**i**])** **%** mod**;**  f**[**0**]** **=** **(**long long**)**f**[**0**]** **\*** n **%** mod**;**  **return** f**;**  **}**  **else**  **{**  int n\_ **=** n **>>** 1**;**  vector**<**int**>** f **=** range\_mul**(**n\_**);**  vector**<**int**>** tmp **=** shift**(**f**,** n\_**);**  f**.**resize**(**n\_ **+** 1**);**  tmp**.**resize**(**n\_ **+** 1**);**  **return** multiply**(**f**,** tmp**);**  **}**  **}**  int f**(**int n**,** int k**)**  **{**  **if** **(**n **==** 0 **&&** k **==** 0**)**  **return** 1**;**  **if** **(**n **<=** 0 **||** k **<=** 0**)**  **return** 0**;**  vector**<**int**>** x **=** vector**<**int**>({**0**,** 1**});**  vector**<**int**>** y **=** range\_mul**(**n **-** 1**);**  vector**<**int**>** ans **=** multiply**(**x**,** y**);**  **if** **(**k **>=** **(**int**)**ans**.**size**())**  **return** 0**;**  **return** ans**[**k**];**  **}**  int ncr**(**int n**,** int r**)**  **{**  **if** **(**r **<** 0 **||** n **<** r**)**  **return** 0**;**  **return** 1LL **\*** fact**[**n**]** **\*** ifact**[**r**]** **%** mod **\*** ifact**[**n **-** r**]** **%** mod**;**  **}**  int main**()**  **{**  fact**[**0**]** **=** 1**;**  **for** **(**int i **=** 1**;** i **<** N**;** **++**i**)**  fact**[**i**]** **=** **(**long long**)**fact**[**i **-** 1**]** **\*** i **%** mod**;**  ifact**[**N **-** 1**]** **=** POW**(**fact**[**N **-** 1**],** mod **-** 2**);**  **for** **(**int i **=** N **-** 1**;** i**;** **--**i**)**  ifact**[**i **-** 1**]** **=** **(**long long**)**ifact**[**i**]** **\*** i **%** mod**;**  int n**,** a**,** b**;**  cin **>>** n **>>** a **>>** b**;**  cout **<<** 1LL **\*** f**(**n **-** 1**,** a **+** b **-** 2**)** **\*** ncr**(**a **+** b **-** 2**,** a **-** 1**)** **%** mod **<<** '\n'**;**  **return** 0**;**  **}**  // <https://codeforces.com/problemset/problem/960/G> |
| |  | | --- | | * *CRT* |   #include <bits/stdc++.h>  **using** **namespace** std**;**  **using** T **=** \_\_int128**;**  // ax + by = \_\_gcd(a, b)  // returns \_\_gcd(a, b)  T extended\_euclid**(**T a**,** T b**,** T **&**x**,** T **&**y**)**  **{**  T xx **=** y **=** 0**;**  T yy **=** x **=** 1**;**  **while** **(**b**)**  **{**  T q **=** a **/** b**;**  T t **=** b**;**  b **=** a **%** b**;**  a **=** t**;**  t **=** xx**;**  xx **=** x **-** q **\*** xx**;**  x **=** t**;**  t **=** yy**;**  yy **=** y **-** q **\*** yy**;**  y **=** t**;**  **}**  **return** a**;**  **}**  // finds x such that x % m1 = a1, x % m2 = a2. m1 and m2 may not be coprime  // here, x is unique modulo m = lcm(m1, m2). returns (x, m). on failure, m = -1.  pair**<**T**,** T**>** CRT**(**T a1**,** T m1**,** T a2**,** T m2**)**  **{**  T p**,** q**;**  T g **=** extended\_euclid**(**m1**,** m2**,** p**,** q**);**  **if** **(**a1 **%** g **!=** a2 **%** g**)**  **return** make\_pair**(**0**,** **-**1**);**  T m **=** m1 **/** g **\*** m2**;**  p **=** **(**p **%** m **+** m**)** **%** m**;**  q **=** **(**q **%** m **+** m**)** **%** m**;**  **return** make\_pair**((**p **\*** a2 **%** m **\*** **(**m1 **/** g**)** **%** m **+** q **\*** a1 **%** m **\*** **(**m2 **/** g**)** **%** m**)** **%** m**,** m**);**  **}**  int32\_t main**()**  **{**  ios\_base**::**sync\_with\_stdio**(**0**);**  cin**.**tie**(**0**);**  cout **<<** **(**int**)**CRT**(**1**,** 31**,** 0**,** 7**).**first **<<** '\n'**;**  **return** 0**;**  **}** |
| |  | | --- | | * *Pollard Rho* |   #include <bits/stdc++.h>  **using** **namespace** std**;**  #define FOR(i, a, b) for (int i = (a), \_##i##\_b = (b); i <= \_##i##\_b; i++)  #define REP(i, a) for (int i = 0, \_##i##\_a = (a); i < \_##i##\_a; i++)  void solve**();**  int32\_t main**()**  **{**  ios**::**sync\_with\_stdio**(**0**);**  cin**.**tie**(**0**);**  solve**();**  **return** 0**;**  **}**  // }}}  // Sieve {{{  // Tested:  // - (up to 5e8) https://j...content-available-to-author-only...o.jp/problem/enumerate\_primes  **typedef** unsigned uint**;**  // NOTE: gP(n) is incorrect for even values of n  const unsigned long long N **=** 2'200'000**;**  uint mark**[**N **/** 64 **+** 1**];**  // DO NOT USE gP(n) directly.  #define gP(n) (mark[(n) >> 6] & (1 << (((n) >> 1) & 31)))  #define rP(n) (mark[(n) >> 6] &= ~(1 << (((n) >> 1) & 31)))  // prime indexed from 0  uint prime**[**200111**],** nprime**;**  void sieve**()**  **{**  memset**(**mark**,** **-**1**,** **sizeof** mark**);**  uint i**;**  uint sqrtN **=** **(**uint**)**sqrt**((**double**)**N**)** **+** 1**;**  **for** **(**i **=** 3**;** i **<** sqrtN**;** i **+=** 2**)**  **if** **(**gP**(**i**))**  **{**  uint i2 **=** i **+** i**;**  **for** **(**uint j **=** i **\*** i**;** j **<** N**;** j **+=** i2**)**  rP**(**j**);**  **}**  nprime **=** 0**;**  prime**[**nprime**++]** **=** 2**;**  **for** **(**i **=** 3**;** i **<** N**;** i **+=** 2**)**  **if** **(**gP**(**i**))**  prime**[**nprime**++]** **=** i**;**  **}**  bool is\_prime\_small**(**uint x**)**  **{**  **if** **(**x **==** 2**)**  **return** **true;**  **if** **(**x **<=** 1**)**  **return** **false;**  **if** **(**x **%** 2 **==** 0**)**  **return** **false;**  **if** **(**gP**(**x**))**  **return** **true;**  **return** **false;**  **}**  // }}}  // Rabin miller {{{  unsigned long long mult**(**unsigned long long x**,** unsigned long long y**,** unsigned long long mod**)**  **{**  **return** \_\_int128\_t**(**x**)** **\*** y **%** mod**;**  **}**  unsigned long long powMod**(**unsigned long long x**,** unsigned long long p**,** unsigned long long mod**)**  **{**  **if** **(**p **==** 0**)**  **return** 1**;**  **if** **(**p **%** 2**)**  **return** mult**(**x**,** powMod**(**x**,** p **-** 1**,** mod**),** mod**);**  **return** powMod**(**mult**(**x**,** x**,** mod**),** p **/** 2**,** mod**);**  **}**  bool checkMillerRabin**(**unsigned long long x**,** unsigned long long mod**,** unsigned long long s**,** int k**)**  **{**  x **=** powMod**(**x**,** s**,** mod**);**  **if** **(**x **==** 1**)**  **return** **true;**  **while** **(**k**--)**  **{**  **if** **(**x **==** mod **-** 1**)**  **return** **true;**  x **=** mult**(**x**,** x**,** mod**);**  **if** **(**x **==** 1**)**  **return** **false;**  **}**  **return** **false;**  **}**  bool is\_prime**(**unsigned long long x**)**  **{**  **if** **(**x **<** N**)**  **return** is\_prime\_small**(**x**);**  REP**(**i**,** 50**)**  **if** **(**x **%** prime**[**i**]** **==** 0**)** **return** **false;**  unsigned long long s **=** x **-** 1**;**  int k **=** 0**;**  **while** **(**s **%** 2 **==** 0**)**  **{**  s **/=** 2**;**  k**++;**  **}**  **if** **(**x **<** 1LL **<<** 32**)**  **{**  **for** **(**unsigned long long z **:** **{**2**,** 7**,** 61**})**  **{**  **if** **(!**checkMillerRabin**(**z**,** x**,** s**,** k**))**  **return** **false;**  **}**  **}**  **else**  **{**  **for** **(**unsigned long long z **:** **{**2**,** 325**,** 9375**,** 28178**})**  **{**  **if** **(!**checkMillerRabin**(**z**,** x**,** s**,** k**))**  **return** **false;**  **}**  **}**  **return** **true;**  **}**  // }}}  vector**<**unsigned long long**>** candidates**;**  // Generate all numbers N = product of at least 3 consecutive primes  const unsigned long long MAX\_VAL **=** 10'000'000'000'000'000'000ULL**;**  void init\_candidates**()**  **{**  **for** **(**unsigned start\_id **=** 0**;** start\_id **<** nprime**;** **++**start\_id**)**  **{**  unsigned long long prod **=** 1LL**;**  **for** **(**unsigned end\_id **=** start\_id**;** end\_id **<** nprime**;** **++**end\_id**)**  **{**  **if** **(**prod **>** MAX\_VAL **/** prime**[**end\_id**])**  **break;**  prod **\*=** prime**[**end\_id**];**  candidates**.**push\_back**(**prod**);**  **}**  **}**  sort**(**candidates**.**begin**(),** candidates**.**end**());**  **}**  unsigned long long safe\_sqrt**(**unsigned long long n**)**  **{**  unsigned long long tmp **=** sqrt**(**n**)** **+** 3**;**  **while** **(**1ULL **\*** tmp **\*** tmp **>** n**)**  tmp**--;**  **return** tmp**;**  **}**  // returns smallest prime >= n  unsigned long long next\_prime**(**unsigned long long n**)**  **{**  **if** **(**n **<=** 2**)**  **return** 2**;**  **if** **(**n **%** 2 **==** 0**)**  **++**n**;**  **while** **(!**is\_prime**(**n**))**  n **+=** 2**;**  **return** n**;**  **}**  unsigned long long prev\_prime**(**unsigned long long n**)**  **{**  assert**(**n **>** 1**);**  **if** **(**n **==** 2**)**  **return** 2**;**  **if** **(**n **<=** 4**)**  **return** 3**;**  **if** **(**n **<=** 6**)**  **return** 5**;**  **while** **(**n **%** 6 **!=** 1 **&&** n **%** 6 **!=** 5**)**  n**--;**  bool stt **=** n **%** 6 **==** 1**;**  **while** **(true)**  **{**  **if** **(**is\_prime**(**n**))**  **return** n**;**  n **-=** stt **?** 2 **:** 4**;**  stt **^=** 1**;**  **}**  **}**  bool check**(**unsigned long long x**)**  **{**  **if** **(**x **==** 1ULL**)**  **return** **false;**  **if** **(**is\_prime**(**x**))**  **return** **true;**  // product of 3 consecutive primes  **if** **(**std**::**binary\_search**(**candidates**.**begin**(),** candidates**.**end**(),** x**))**  **return** **true;**  auto next **=** next\_prime**(**safe\_sqrt**(**x**)** **+** 1**);**  **if** **(**x **%** next**)**  **return** **false;**  **if** **(!**is\_prime**(**x **/** next**))**  **return** **false;**  // product of 2 consecutive primes  **return** x **==** next **\*** prev\_prime**(**next **-** 1**);**  **}**  void solve**()**  **{**  sieve**();**  init\_candidates**();**  int ntest**;**  cin **>>** ntest**;**  **while** **(**ntest**--)**  **{**  unsigned long long x**;**  cin **>>** x**;**  cout **<<** **(**check**(**x**)** **?** "NICE" **:** "UGLY"**)** **<<** '\n'**;**  **}**  **}** |
| |  | | --- | | * *Bitmask* |   #define MASK(i) (1 << (i)) // make mask  #define COUNT\_BIT(x) \_\_builtin\_popcount(x) // counting set-on bits of x  #define STATE(x, i) ((x) & (1 << (i))) // state of ith bit of x  #define SET\_ON(x, i) ((x) | (1 << (i))) // set the ith bit of x on  #define SET\_OFF(x, i) ((x) & ~(1 << (i))) // set the ith bit of x off  #define LEAST\_BIT(x) ((x) & (-x)) |
| |  | | --- | | * *Catalan* |   Mở ảnh |
| |  | | --- | | * *Phi hàm euler* |   ll e**[**N**];**  void init**()**  **{**  For**(**i**,** 1**,** N **-** 1**)** e**[**i**]** **=** i**;**  For**(**i**,** 2**,** N **-** 1**){**  **if(**e**[**i**]** **==** i**){**  e**[**i**]** **=** i **-** 1**;**  **for** **(**ll j **=** i **\*** 2**;** j **<** N**;** j **+=** i**)**  e**[**j**]** **=** e**[**j**]** **-** e**[**j**]** **/** i**;**  **}**  **}**  **}** |
| |  | | --- | | * *Extend euclid* |   int gcd**(**int a**,** int b**,** int **&**x**,** int **&**y**)**  **{**  **if** **(**b **==** 0**)**  **{**  x **=** 1**;**  y **=** 0**;**  **return** a**;**  **}**  int x1**,** y1**;**  int d **=** gcd**(**b**,** a **%** b**,** x1**,** y1**);**  x **=** y1**;**  y **=** x1 **-** y1 **\*** **(**a **/** b**);**  **return** d**;**  **}**  void solution**()**  **{**  int x**,** y**;**  int g **=** gcd**(**a**,** mod**,** x**,** y**);**  **if** **(**g **!=** 1**)**  **{**  cout **<<** "No solution!"**;**  **}**  **else**  **{**  x **=** **(**x **%** MOD **+** MOD**)** **%** MOD**;**  cout **<<** x **<<** "\n"**;**  **}**  **}** |
| |  | | --- | | * *Công thức tính sum* |   Không có mô tả. |
| |  | | --- | | * *Tính tổng các số nguyên tố nhỏ hơn hoặc bằng n* |   #include <iostream>  #include <vector>  #include <cmath>  #include <chrono>  **using** **namespace** std**;**  **using** **namespace** std**::**chrono**;**  long long S**(**long long N**)**  **{**  long long r **=** **(**long long**)**sqrt**(**N**);**  vector**<**long long**>** a**(**r **+** 1**);**  vector**<**long long**>** b**(**r **+** 1**);**  **for** **(**long long i **=** 1**;** i **<=** r**;** i**++)**  **{**  a**[**i**]** **=** i **\*** **(**i **+** 1**)** **/** 2 **-** 1**;**  b**[**i**]** **=** **(**N **/** i**)** **\*** **(**N **/** i **+** 1**)** **/** 2 **-** 1**;**  **}**  **for** **(**long long p **=** 2**;** p **<=** r**;** p**++)**  **if** **(**a**[**p**]** **>** a**[**p **-** 1**])**  **{**  long long sp **=** a**[**p **-** 1**];**  long long p2 **=** p **\*** p**;**  long long to **=** min**(**r**,** N **/** p2**);**  **for** **(**long long i **=** 1**;** i **<=** to**;** i**++)**  **{**  long long vp **=** i **\*** p**;**  **if** **(**vp **<=** r**)**  vp **=** b**[**vp**];**  **else**  vp **=** a**[**N **/** vp**];**  b**[**i**]** **-=** p **\*** **(**vp **-** sp**);**  **}**  **for** **(**long long v **=** r**;** v **>=** p2**;** v**--)**  a**[**v**]** **-=** p **\*** **(**a**[**v **/** p**]** **-** sp**);**  **}**  **return** b**[**1**];**  **}**  int main**()**  **{**  long long n**;**  cin **>>** n**;**  cout **<<** S**(**n**)** **<<** endl**;**  **}** |

# **3. Hình học**

|  |
| --- |
| #define x first  #define y second  #define pi 3.14159265359  typedef pair<double, double> ii; |
| |  | | --- | | * *Hình tròn* |   struct Circle  **{**      double x**,** y**,** r**;**      ii O{x,y}**;**      double area**()**  **{**  **return** r **\*** r **\*** pi**;**  **}**      double sector\_area**(**double theta**)**  **{**  **return** 0.5 **\*** r **\*** r **\*** theta**;**  **}**    **};** |
| |  | | --- | | * *Đường thẳng* |   struct Line  **{**  ii A**,**B**;**  double a**,**b**,**c**;**  Line**(**ii A**,** ii B**)**  **{**  **this->**A **=** A**;**  **this->**B **=** B**;**  b **=** **(**A**.**x **-** B**.**x**);**  a **=** **(**B**.**y **-** A**.**y**);**  c **=** **-(**a **\*** A**.**x **+** b **\*** A**.**y**);**  **}**  Line**(**double valuea**,** double valueb**,** double valuec**)**  **{**  a **=** valuea**;**  b **=** valueb**;**  c **=** valuec**;**  **}**  double length**()**  **{**  **return** sqrt**(** **(**a **\*** a**)** **+** **(**b **\*** b**)** **);**  **}**  **};**    // Tích vô hướng vector AB nhân vector AC  double dot**(**ii A**,** ii B**,** ii C**)** **{**  ii AB**,** AC**;**  AB**.**x **=** B**.**x **-** A**.**x**;**  AB**.**y **=** B**.**y **-** A**.**y**;**  AC**.**x **=** C**.**x **-** A**.**x**;**  AC**.**y **=** C**.**y **-** A**.**y**;**  **return** AB**.**x **\*** AC**.**x **+** AB**.**y **\*** AC**.**y**;**  **}**  // cross (used for Convex Hull and ccw)  double cross**(**ii A**,** ii B**,** ii C**)** **{**    **return** **(**B**.**x **-** A**.**x**)** **\*** **(**C**.**y **-** A**.**y**)** **-** **(**B**.**y **-** A**.**y**)** **\*** **(**C**.**x **-** A**.**x**);**  **}**    int ccw**(**ii A**,** ii B**,** ii C**)**  **{**  double area2 **=** cross**(**A**,** B**,** C**);**  **if(**area2 **<** 0**)**  **{**  // clock\_wise  **return** **-**1**;**  **}**  **if(**area2 **>** 0**)**  **{**  //counter\_clock\_wise  **return** 1**;**  **}**  **if(**area2 **==** 0**)**  **{**  //collinear  **return** 0**;**  **}**  **}**    //tinh goc giua 2 vecto  double cos**(**ii A**,** ii B**,** ii C**)**  **{**  //maybe <0 -> maybe corner > pi/2  **return** dot**(**A**,**B**,**C**)** **/** distance**(**A**,**B**)** **/** distance**(**A**,**C**);**  **}**  // tính d(A,B)  double distance**(**ii A**,** ii B**)** **{**  int dx **=** A**.**x **-** B**.**x**;**  int dy **=** A**.**y **-** B**.**y**;**  **return** sqrt**(**dx **\*** dx **+** dy **\*** dy**);**  **}**  // tính d(AB,C)  double distance**(**ii A**,** ii B**,** ii C**)** **{**  int dx **=** A**.**x **-** B**.**x**;**  int dy **=** A**.**y **-** B**.**y**;**  **return** sqrt**(**dx **\*** dx **+** dy **\*** dy**);**  **}**  double distance**(**Line AB**,** ii C**)** **{**  int dx **=** A**.**x **-** B**.**x**;**  int dy **=** A**.**y **-** B**.**y**;**  **return** sqrt**(**dx **\*** dx **+** dy **\*** dy**);**  **}**  // AB là đoạn thẳng nếu isSegment=true  // AB là đường thẳng nếu isSegment=false  double linePointDist**(**ii A**,** ii B**,** ii C**,** bool isSegment**)** **{**  double dist **=** abs**(**cross**(**A**,** B**,** C**))** **/** distance**(**A**,** B**);**  **if** **(**isSegment**)** **{**  int dot1 **=** dot**(**B**,** A**,** C**);**  **if** **(**dot1 **<** 0**)** **return** distance**(**B**,** C**);**  int dot2 **=** dot**(**A**,** B**,** C**);**  **if** **(**dot2 **<** 0**)** **return** distance**(**A**,** C**);**  **}**  **return** dist**;**  **}**    //Giao điểm 2 đường thẳng A1x+B1y=C1 và A2x+B2y=C2:  int checkIntersection**(**Line A**,** Line B**,** ii **&**M**)**  **{**  // double det = A1 \* B2 - A2 \* B1;  double det **=** A**.**a **\*** B**.**b **-** A**.**b **\*** B**.**a**;**  **if** **(**det **==** 0**)**  **{**  // Lines are parallel or coincident  // if (A1 \* C2 == A2 \* C1)  **if** **(**A**.**a **\*** B**.**c **==** B**.**a **\*** A**.**c**)**  **{**  // Lines are coincident  //return -1  **return** **-**1**;**  **}**  **else**  **{**  // Lines are parallel  //return 0  **return** 0**;**  **}**  **}**  **else**  **{**  //return 1  double x **=** **(**B**.**b **\*** A**.**c **-** A**.**b **\*** B**.**c**)** **/** det**;**  //xu ly sai so double khi x = -0  **if(**x **==** **-**0**)** x **=** 0**;**    double y **=** **(**A**.**a **\*** B**.**c **-** B**.**a **\*** A**.**c**)** **/** det**;**  **if(**y **==** **-**0**)** y **=** 0**;**  M **=** **{**x**,** y**};**  **return** 1**;**  **}**    **}**    double distance**(**ii A**,** ii B**,** ii C**)** **{**  Line AB**={**A**,**B**};**  **return** abs**(**AB**.**a **\*** C**.**x **+** AB**.**b **\*** C**.**y **+** AB**.**c**)** **/** AB**.**length**();**  **}**    double distance**(**Line AB**,** ii C**)** **{**  **return** abs**(**AB**.**a **\*** C**.**x **+** AB**.**b **\*** C**.**y **+** AB**.**c**)** **/** AB**.**length**();**  **}**  //tinh goc giua 2 duong thang      double cos2**(**ii A**,** ii B**,** ii C**)**  **{**  //always >0 -> corner <= pi/2;  Line AB **=** **{**A**,**B**},** AC **=** **{**A**,**C**};**  **return** abs**(**AB**.**a **\*** AC**.**a **+** AB**.**b **\*** AC**.**b**)** **/** AB**.**length**()** **/** AC**.**length**();**  **}**    //kiem tra 2 diem A,B co cung nam ve 1 phia so voi ab  bool check**(**ii A**,** ii B**,** Line ab**)**  **{**  double a **=** ab**.**a **\*** A**.**x **+** ab**.**b **\*** A**.**y **+** ab**.**c**;**  double b **=** ab**.**a **\*** B**.**x **+** ab**.**b **\*** B**.**y **+** ab**.**c**;**  **if(**a **\*** b **>** 0**)**  **{**  **return** **true;**  **}**  **return** **false;**  **}** |
| |  | | --- | | * *Tam giác* |   struct Triangle  **{**      ii A**,**B**,**C**;**      double a**,**b**,**c**;**      Triangle**()**  **{**          A **=** **{**0**,**0**},** B **=** **{**0**,**0**},** C **=** **{**0**,**0**};**          a **=** 0**,** b **=** 0**,** c **=** 0**;**  **}**      Triangle **(**ii A **,** ii B**,** ii C**)**  **{**          Line AB**{**A**,**B**},** BC**{**B**,**C**},** AC**{**A**,**C**};**           a **=** BC**.**length**(),** b **=** AC**.**length**(),** c **=** AB**.**length**();**  **}**      double perimeter**()**  **{**  **return** **(**a**+**b**+**c**)/**2**;**  **}**      double area**()**  **{**          double p **=** **this->**perimeter**();**  **return** sqrt**(**p **\*** **(**p **-** a**)** **\*** **(**p **-** b**)** **\*** **(**p **-** c**));**  **}**  **}** |
| |  | | --- | | * *Bao lồi* |   ii p**[**105**]** **=** **{};** // p điểm  ii poly**[**105**]** **={};** // bao lồi  void ConvexHull**()**  **{**  sort**(**p**+**1**,** p**+**1**+**n**);**  int k **=** 0**;**  FOR**(**i**,**1**,**n**)**  **{**  **while(** k **>=** 2 **&&** cross**(**poly**[**k**-**2**],** poly**[**k**-**1**],** p**[**i**])** **<=** 0**)**  **{**  k**--;**  **}**  poly**[**k**++]** **=** p**[**i**];**  **}**  **for(**int i **=** n **-** 1**,** t **=** k **+** 1**;** i **>=** 1**;** i**--)**  **{**  **while(**k **>=** t **&&** cross**(**poly**[**k**-**2**],** poly**[**k**-**1**],** p**[**i**])** **<=** 0**)**  **{**  k**--;**  **}**  poly**[**k**++]** **=** p**[**i**];**  **}**  **}** |
| |  | | --- | | * *Diện tích đa giác* |   double area**(**ii poly**[],** int Size**)**  **{**  //poly index from 0 to Size  poly**[**Size**+**1**]** **=** poly**[**0**];**  double S **=** **(**poly**[**Size**+**1**].**x **-** poly**[**0**].**x**)** **\*** **(**poly**[**Size**+**1**].**y **+** poly**[**0**].**y**);**  FOR**(**i**,** 1**,** Size**+**1**)**  **{**  S **+=** **(**poly**[**i**-**1**].**x **-** poly**[**i**].**x**)** **\*** **(**poly**[**i**-**1**].**y **+** poly**[**i**].**y**);**  **}**  **return** abs**(**S**)/**2**;**  **}** |
| |  | | --- | | * *Kiểm tra 1 điểm nằm trong đa giác* |   //on edge -> false, index from 1 to Size  bool check**(**ii A**,** ii poly**[],** int Size**)**  **{**  double sum **=** 0**;**  poly**[**n**+**1**]** **=** poly**[**1**];**  **for(**int i **=** 1**;** i **<=** Size**;** i**++)**  **{**  int pv **=** 0**;**  **if(**cross**(**A**,**poly**[**i**],** poly**[**i**+**1**])** **>** 0**)** pv **=** 1**;**  **else** **if(**cross**(**A**,** poly**[**i**],** poly**[**i**+**1**])** **<** 0**)** pv **=** **-**1**;**  sum **=** sum **+** acos**(**cos**(**A**,**poly**[**i**],** poly**[**i**+**1**]))** **\*** pv**;**  **}**  **if(**abs**(**sum **-** 2**\***pi**)** **<=** 0.00001**)** **return** **true;**  **return** **false;**    **}** |

# **4.Data Structures**

|  |  |
| --- | --- |
| |  | | --- | | * *Segment Tree* |   struct Segment  **{**  v seg**,** lazy**;**  Segment**(**int n**)**  **{**  seg**.**resize**(**4 **\*** n**,** 0**);**  lazy**.**resize**(**4 **\*** n**,** 0**);**  **}**  int merge**(**int x**,** int y**)**  **{**  **return** x **+** y**;**  **}**  int add**(**int l**,** int r**,** int x**)**  **{**  **return** **(**r **-** l **+** 1**)** **\*** x**;**  // return x;  **}**  void down**(**int id**,** int l**,** int r**)**  **{**  int t **=** lazy**[**id**];**  lazy**[**id**]** **=** 0**;**  int m **=** **(**l **+** r**)** **>>** 1**;**  seg**[**id **<<** 1**]** **+=** add**(**l**,** m**,** t**);**  lazy**[**id **<<** 1**]** **+=** t**;**  seg**[**id **<<** 1 **|** 1**]** **+=** add**(**m **+** 1**,** r**,** t**);**  lazy**[**id **<<** 1 **|** 1**]** **+=** t**;**  **}**  void update**(**int id**,** int l**,** int r**,** int u**,** int v**,** int val**)**  **{**  **if** **(**l **>** v **||** r **<** u **||** l **>** r**)**  **return;**  **if** **(**l **>=** u **&&** r **<=** v**)**  **{**  seg**[**id**]** **+=** add**(**l**,** r**,** val**);**  lazy**[**id**]** **+=** val**;**  **return;**  **}**  down**(**id**,** l**,** r**);**  int mid **=** **(**l **+** r**)** **>>** 1**;**  update**(**id **<<** 1**,** l**,** mid**,** u**,** v**,** val**);**  update**(**id **<<** 1 **|** 1**,** mid **+** 1**,** r**,** u**,** v**,** val**);**  seg**[**id**]** **=** merge**(**seg**[**id **<<** 1**],** seg**[**id **<<** 1 **|** 1**]);**  **}**  int get**(**int id**,** int l**,** int r**,** int u**,** int v**)**  **{**  **if** **(**l **>** v **||** r **<** u **||** l **>** r**)**  **return** 0**;**  **if** **(**l **>=** u **&&** r **<=** v**)**  **{**  **return** seg**[**id**];**  **}**  down**(**id**,** l**,** r**);**  int mid **=** **(**l **+** r**)** **>>** 1**;**  int v1 **=** get**(**id **<<** 1**,** l**,** mid**,** u**,** v**);**  int v2 **=** get**(**id **<<** 1 **|** 1**,** mid **+** 1**,** r**,** u**,** v**);**  **return** merge**(**v1**,** v2**);**  **}**  void build**(**int a**[],** int id**,** int l**,** int r**)**  **{**  **if** **(**l **==** r**)**  **{**  seg**[**id**]** **=** a**[**l**];**  **return;**  **}**  int mid **=** **(**l **+** r**)** **>>** 1**;**  build**(**a**,** id **<<** 1**,** l**,** mid**);**  build**(**a**,** id **<<** 1 **|** 1**,** mid **+** 1**,** r**);**  seg**[**id**]** **=** merge**(**seg**[**id **<<** 1**],** seg**[**id **<<** 1 **|** 1**]);**  **}**  **};** |
| |  | | --- | | * *Trie* |   struct TrieNode  **{**  struct TrieNode **\***children**[**SIZE**];**  int f**[**SIZE**];**  bool isEnd**;**  **};**    struct TrieNode **\***getNode**(**void**)** // Tao mot nut moi  **{**  struct TrieNode **\***pNode **=** **new** TrieNode**;**  pNode**->**isEnd **=** **false;**  **for** **(**int i **=** 0**;** i **<** SIZE**;** i**++)**  **{**  pNode**->**children**[**i**]** **=** **NULL;**  pNode**->**f**[**i**]** **=** 0**;**  **}**  **return** pNode**;**  **}**    void Insert**(**struct TrieNode **\***root**,** string key**)** // Chen xau vao  **{**  struct TrieNode **\***parent **=** root**;**  **for** **(**int i **=** 0**;** i **<** key**.**size**();** i**++)**  **{**  int id **=** key**[**i**]** **-** 'a'**;**  **if** **(!**parent**->**children**[**id**])**  parent**->**children**[**id**]** **=** getNode**();**  parent**->**f**[**id**]++;**  parent **=** parent**->**children**[**id**];**  **}**  parent**->**isEnd **=** **true;**  **}**    int search**(**struct TrieNode **\***root**,** string key**)** // Tim kiem  **{**  int res**;**  struct TrieNode **\***parent **=** root**;**  **for** **(**int i **=** 0**;** i **<** key**.**size**();** i**++)**  **{**  int id **=** key**[**i**]** **-** 'a'**;**  **if** **(!**parent**->**children**[**id**])**  **return** **false;**  res **=** parent**->**f**[**id**];**  parent **=** parent**->**children**[**id**];**  **}**  **return** res**;**  **}**    void PrintWords**(**struct TrieNode **\***root**,** char str**[],** int id**)**  **{**  **if** **(**root**->**isEnd**)**  **{**  str**[**id**]** **=** '\0'**;**  cout **<<** str **<<** '\n'**;**  **}**    **for** **(**int i **=** 0**;** i **<** SIZE**;** i**++)**  **if** **(**root**->**children**[**i**])**  **{**  str**[**id**]** **=** i **+** 'a'**;**  PrintWords**(**root**->**children**[**i**],** str**,** id **+** 1**);**  **}**  **}**    bool isEmpty**(**struct TrieNode **\***root**)**  **{**  **for** **(**int i **=** 0**;** i **<** SIZE**;** i**++)**  **if** **(**root**->**children**[**i**])**  **return** **false;**  **return** **true;**  **}**    void Del**(**struct TrieNode **\***root**,** string key**,** int level**)**  **{**  **if** **(**level **==** key**.**length**())** **return;**  int id **=** key**[**level**]** **-** 'a'**;**  Del**(**root**->**children**[**id**],** key**,** level **+** 1**);**  **if** **(**isEmpty**(**root**->**children**[**id**]))**  **{**  **delete** root**->**children**[**id**];**  root**->**children**[**id**]** **=** **NULL;**  **}**  **}**    struct TrieNode **\***root **=** getNode**();** |
| |  | | --- | | * *Ordered* |   #include <ext/pb\_ds/assoc\_container.hpp>  #include <ext/pb\_ds/tree\_policy.hpp>  #include <ext/pb\_ds/detail/standard\_policies.hpp>  #pragma GCC optimize("O3")  #pragma GCC target("sse4")  **using** **namespace** \_\_gnu\_pbds**;**  template **<**typename T**>**  **using** ordered\_set **=** tree**<**T**,** null\_type**,** less**<**T**>,** rb\_tree\_tag**,** tree\_order\_statistics\_node\_update**>;**  template **<**typename T**>**  **using** ordered\_multiset **=** tree**<**T**,** null\_type**,** less\_equal**<**T**>,** rb\_tree\_tag**,** tree\_order\_statistics\_node\_update**>;** |
| |  | | --- | | * *Segment Tree (ver2)* |   struct Seg  **{**  int BUF **=** 1000005**;**  int val**[**2000010**],** lazy**[**2000010**];**  void get**(**int i**)** **{** val**[**i**]** **=** max**(**val**[**i **<<** 1**],** val**[**i **<<** 1 **|** 1**])** **+** lazy**[**i**];** **}**  void update**(**int i**,** int j**,** int k**)**  **{** //[i,j]  i **+=** BUF **-** 1**,** j **+=** BUF **+** 1**;** //(i,j)  **while** **(**i **|** j**)**  **{**  **if** **(**j **-** i **>** 1**)**  **{**  **if** **(~**i **&** 1**)**  val**[**i **^** 1**]** **+=** k**,** lazy**[**i **^** 1**]** **+=** k**;**  **if** **(**j **&** 1**)**  val**[**j **^** 1**]** **+=** k**,** lazy**[**j **^** 1**]** **+=** k**;**  **}**  i **>>=** 1**,** j **>>=** 1**;**  get**(**i**),** get**(**j**);**  **}**  **}**  int query**(**int i**,** int j**)**  **{** //[i,j]  i **+=** BUF **-** 1**,** j **+=** BUF **+** 1**;** //(i,j)  int resl **=** **-**1e18**,** resr **=** **-**1e18**;**  **while** **(**i **|** j**)**  **{**  **if** **(**j **-** i **>** 1**)**  **{**  **if** **(~**i **&** 1**)**  resl **=** max**(**resl**,** val**[**i **^** 1**]);**  **if** **(**j **&** 1**)**  resr **=** max**(**resr**,** val**[**j **^** 1**]);**  **}**  i **>>=** 1**,** j **>>=** 1**;**  resl **+=** lazy**[**i**],** resr **+=** lazy**[**j**];**  **}**  **return** max**(**resl**,** resr**);**  **}**  **};** |
| |  | | --- | | * *SegmentTree ver3* |   struct node\_3 **{**  int TREE**[**N **\*** 2**];**  int range**[**N **\*** 2**];**  int lazy**[**N**];**  int h**;**    node\_3**()** **{** refresh**();** **}**    void refresh**()** **{**  h **=** **sizeof(**int**)** **\*** 8 **-** \_\_builtin\_clz**(**n**);**  FOR**(**i**,** 0**,** n**-**1**)** lazy**[**i**]** **=** 0**;**  FOR**(**i**,** 0**,** 2**\***n**-**1**)** TREE**[**i**]** **=** 0**;**    FOR**(**i**,** 0**,** n **-** 1**)** **{**  range**[**i **+** n**]** **=** 1**;**  **}**    FORD**(**i**,** n **-** 1**,** 1**)** **{**  range**[**i**]** **=** range**[**i **<<** 1**]** **+** range**[**i **<<** 1 **|** 1**];**  **}**  **}**    void build**(**ll idx**)**  **{**  **while** **(**idx **>>=** 1**)**  **{**  TREE**[**idx**]** **=** lazy**[**idx**]** **\*** range**[**idx**]** **+** TREE**[**idx **<<** 1**]** **+** TREE**[**idx **<<** 1 **|** 1**];**  **}**  **}**    void update**(**ll l**,** ll r**,** ll num**)**  **{**  ll l0 **=** l **+=** n**,** r0 **=** r **+=** n**;**  **for** **(**r**++;** l **<** r**;** l **>>=** 1**,** r **>>=** 1**)**  **{**  **if** **(**l **&** 1**)** apply**(**l**++,** num**);**  **if** **(**r **&** 1**)** apply**(--**r**,** num**);**  **}**  build**(**l0**);** build**(**r0**);**  **}**    void apply**(**ll idx**,** ll num**)**  **{**  TREE**[**idx**]** **+=** num **\*** range**[**idx**];**  **if** **(**idx **<** n**)** lazy**[**idx**]** **+=** num**;**  **}**    void push**(**ll idx**)**  **{**  **for** **(**int i **=** h**;** i **>** 0**;** **--**i**)** **{**  int k **=** idx **>>** i**;**  **if** **(**lazy**[**k**]** **!=** 0**)** **{**  apply**(**k **<<** 1**,** lazy**[**k**]);**  apply**(**k **<<** 1 **|** 1**,** lazy**[**k**]);**  **}**  lazy**[**k**]** **=** 0**;**  **}**  **}**    ll query**(**ll l**,** ll r**)**  **{**  int ans **=** 0**;**  push**(**l **+=** n**),** push**(**r **+=** n**);**  **for** **(**r**++;** l **<** r**;** l **>>=** 1**,** r **>>=** 1**)** **{**  **if** **(**l**&**1**)** ans **+=** TREE**[**l**++];**  **if** **(**r**&**1**)** ans **+=** TREE**[--**r**];**  **}**  **return** ans**;**  **}**  **};** |
| |  | | --- | | * *BIT 2D* |   struct bit2d  **{**  ll a**[**N**][**N**][**2**],** b**[**N**][**N**][**2**];**  bit2d**()**  **{**  memset**(**a**,** 0**,** **sizeof(**a**));**  memset**(**b**,** 0**,** **sizeof(**b**));**  **}**  void update2**(**ll t**[**N**][**N**][**2**],** ll x**,** ll y**,** ll mul**,** ll add**)**  **{**  **for** **(**ll i **=** x**;** i **<** N**;** i **+=** i **&** **-**i**)**  **{**  **for** **(**ll j **=** y**;** j **<** N**;** j **+=** j **&** **-**j**)**  **{**  t**[**i**][**j**][**0**]** **+=** mul**;**  t**[**i**][**j**][**1**]** **+=** add**;**  **}**  **}**  **}**  void update1**(**ll x**,** ll y1**,** ll y2**,** ll mul**,** ll add**)**  **{**  update2**(**a**,** x**,** y1**,** mul**,** **-**mul **\*** **(**y1 **-** 1**));**  update2**(**a**,** x**,** y2**,** **-**mul**,** mul **\*** y2**);**  update2**(**b**,** x**,** y1**,** add**,** **-**add **\*** **(**y1 **-** 1**));**  update2**(**b**,** x**,** y2**,** **-**add**,** add **\*** y2**);**  **}**  void update**(**ll x1**,** ll y1**,** ll x2**,** ll y2**,** ll val**)**  **{**  update1**(**x1**,** y1**,** y2**,** val**,** **-**val **\*** **(**x1 **-** 1**));**  update1**(**x2**,** y1**,** y2**,** **-**val**,** val **\*** x2**);**  **}**  ll query2**(**ll t**[**N**][**N**][**2**],** int x**,** int y**)**  **{**  ll mul **=** 0**,** add **=** 0**;**  **for** **(**int i **=** y**;** i **>** 0**;** i **-=** i **&** **-**i**)**  **{**  mul **+=** t**[**x**][**i**][**0**];**  add **+=** t**[**x**][**i**][**1**];**  **}**  **return** mul **\*** x **+** add**;**  **}**  ll query1**(**int x**,** int y**)**  **{**  ll mul **=** 0**,** add **=** 0**;**  **for** **(**int i **=** x**;** i **>** 0**;** i **-=** i **&** **-**i**)**  **{**  mul **+=** query2**(**a**,** i**,** y**);**  add **+=** query2**(**b**,** i**,** y**);**  **}**  **return** mul **\*** x **+** add**;**  **}**  ll query**(**int x1**,** int y1**,** int x2**,** int y2**)**  **{**  **return** query1**(**x2**,** y2**)** **-** query1**(**x1 **-** 1**,** y2**)** **-** query1**(**x2**,** y1 **-** 1**)** **+** query1**(**x1 **-** 1**,** y1 **-** 1**);**  **}**  **}** t**;** |
| |  | | --- | | * *MO* |   void remove**(**idx**);** // TODO: remove value at idx from data structure  void add**(**idx**);** // TODO: add value at idx from data structure  int get\_answer**();** // TODO: extract the current answer of the data structure int block\_size;  struct Query **{**  int l**,** r**,** idx**;**  bool **operator<(**Query other**)** const**{**  **return** make\_pair**(**l **/** block\_size**,** r**)** **<** make\_pair**(**other**.**l **/** block\_size**,** other**.**r**);**  **}** **};**  vector**<**int**>** mo\_algorithm**(**vector**<**Query**>** queries**)** **{**  vector**<**int**>** answers**(**queries**.**size**());** sort**(**queries**.**begin**(),** queries**.**end**());** /\*Optimize 1  sort(queries.begin(), queries.end(), [](node a, node b){  return (a.l / blockSize < b.l / blockSize || (a.l / blockSize == b.l / blockSize  && a.r < b.r));  });  bool cmp(pair<int, int> p, pair<int, int> q) {  if (p.first / BLOCK\_SIZE != q.first / BLOCK\_SIZE)  q.second); }  \*/  **return** p **<** q**;**  **return** **(**p**.**first **/** BLOCK\_SIZE **&** 1**)** **?** **(**p**.**second **<** q**.**second**)** **:** **(**p**.**second **>**  int cur\_l **=** 0**,** cur\_r **=** **-**1**;**  // invariant: data structure will always reflect the range [cur\_l, cur\_r] for (Query q : queries) {  **while** **(**cur\_l **>** q**.**l**)** **{**  cur\_l**--;**  add**(**cur\_l**);**  **}**  **while** **(**cur\_r **<** q**.**r**)** **{**  cur\_r**++;**  add**(**cur\_r**);**  **}**  **while** **(**cur\_l **<** q**.**l**)** **{**  remove**(**cur\_l**);**  cur\_l**++;** **}**  **while** **(**cur\_r **>** q**.**r**)** **{**  remove**(**cur\_r**);**  cur\_r**--;** **}**  answers**[**q**.**idx**]** **=** get\_answer**();** **}**  **return** answers**;** |
| |  | | --- | | * *Squareroot decomposition* |   //return number of elements equal to K  const int BLOCK\_SIZE **=** 320**;** const int N **=** 1e5 **+** 2**;**  int n**;**  int cnt**[**N **/** BLOCK\_SIZE **+** 2**][**N**];** int a**[**N**];**  void preprocess**()**  **{**  **for** **(**int i **=** 0**;** i **<** n**;** **++**i**)** **++**cnt**[**i **/** BLOCK\_SIZE**][**a**[**i**]];**  **}**  int query**(**int l**,** int r**,** int k**)** **{**  int blockL **=** **(**l **+** BLOCK\_SIZE **-** 1**)** **/** BLOCK\_SIZE**;** int blockR **=** r **/** BLOCK\_SIZE**;**  **if** **(**blockL **>=** blockR**)**  **return** count**(**a **+** l**,** a **+** r **+** 1**,** k**);** // using stl  int sum **=** 0**;**  **for** **(**int i **=** blockL**;** i **<** blockR**;** **++**i**)**  sum **+=** cnt**[**i**][**k**];**  **for** **(**int i **=** l**,** lim **=** blockL **\*** BLOCK\_SIZE**;** i **<** lim**;** **++**i**)**  **if** **(**a**[**i**]** **==** k**)** **++**sum**;**  **for** **(**int i **=** blockR **\*** BLOCK\_SIZE**;** i **<=** r**;** **++**i**)**  **if** **(**a**[**i**]** **==** k**)** **++**sum**;**  **return** sum**;**  **}**  void update**(**int u**,** int v**)**  **{**  int block **=** u **/** BLOCK\_SIZE**;**  **--**cnt**[**block**][**a**[**u**]];**  a**[**u**]** **=** v**;**  **++**cnt**[**block**][**a**[**u**]];**  **}** |

# **5. Graph**

|  |  |
| --- | --- |
| |  | | --- | | * *2sat* |   #include <bits/stdc++.h>  **using** **namespace** std**;**  const int N **=** 3e5 **+** 9**;**  /\*  zero Indexed  we have vars variables  F=(x\_0 XXX y\_0) and (x\_1 XXX y\_1) and ... (x\_{vars-1} XXX y\_{vars-1})  here {x\_i,y\_i} are variables  and XXX belongs to {OR,XOR}  is there any assignment of variables such that F=true  \*/  struct twosat  **{**  int n**;** // total size combining +, -. must be even.  vector**<**vector**<**int**>>** g**,** gt**;**  vector**<**bool**>** vis**,** res**;**  vector**<**int**>** comp**;**  stack**<**int**>** ts**;**  twosat**(**int vars **=** 0**)**  **{**  n **=** vars **<<** 1**;**  g**.**resize**(**n**);**  gt**.**resize**(**n**);**  **}**  // zero indexed, be careful  // if you want to force variable a to be true in OR or XOR combination  // add addOR (a,1,a,1);  // if you want to force variable a to be false in OR or XOR combination  // add addOR (a,0,a,0);  //(x\_a or (not x\_b))-> af=1,bf=0  void addOR**(**int a**,** bool af**,** int b**,** bool bf**)**  **{**  a **+=** a **+** **(**af **^** 1**);**  b **+=** b **+** **(**bf **^** 1**);**  g**[**a **^** 1**].**push\_back**(**b**);** // !a => b  g**[**b **^** 1**].**push\_back**(**a**);** // !b => a  gt**[**b**].**push\_back**(**a **^** 1**);**  gt**[**a**].**push\_back**(**b **^** 1**);**  **}**  //(!x\_a xor !x\_b)-> af=0, bf=0  void addXOR**(**int a**,** bool af**,** int b**,** bool bf**)**  **{**  addOR**(**a**,** af**,** b**,** bf**);**  addOR**(**a**,** **!**af**,** b**,** **!**bf**);**  **}**  // add this type of condition->  // add(a,af,b,bf) means if a is af then b must need to be bf  void add**(**int a**,** bool af**,** int b**,** bool bf**)**  **{**  a **+=** a **+** **(**af **^** 1**);**  b **+=** b **+** **(**bf **^** 1**);**  g**[**a**].**push\_back**(**b**);**  gt**[**b**].**push\_back**(**a**);**  **}**  void dfs1**(**int u**)**  **{**  vis**[**u**]** **=** **true;**  **for** **(**int v **:** g**[**u**])**  **if** **(!**vis**[**v**])**  dfs1**(**v**);**  ts**.**push**(**u**);**  **}**  void dfs2**(**int u**,** int c**)**  **{**  comp**[**u**]** **=** c**;**  **for** **(**int v **:** gt**[**u**])**  **if** **(**comp**[**v**]** **==** **-**1**)**  dfs2**(**v**,** c**);**  **}**  bool ok**()**  **{**  vis**.**resize**(**n**,** **false);**  **for** **(**int i **=** 0**;** i **<** n**;** **++**i**)**  **if** **(!**vis**[**i**])**  dfs1**(**i**);**  int scc **=** 0**;**  comp**.**resize**(**n**,** **-**1**);**  **while** **(!**ts**.**empty**())**  **{**  int u **=** ts**.**top**();**  ts**.**pop**();**  **if** **(**comp**[**u**]** **==** **-**1**)**  dfs2**(**u**,** scc**++);**  **}**  res**.**resize**(**n **/** 2**);**  **for** **(**int i **=** 0**;** i **<** n**;** i **+=** 2**)**  **{**  **if** **(**comp**[**i**]** **==** comp**[**i **+** 1**])**  **return** **false;**  res**[**i **/** 2**]** **=** **(**comp**[**i**]** **>** comp**[**i **+** 1**]);**  **}**  **return** **true;**  **}**  **};**  int main**()**  **{**  int n**,** m**;**  cin **>>** n **>>** m**;**  twosat ts**(**n**);**  **for** **(**int i **=** 0**;** i **<** m**;** i**++)**  **{**  int u**,** v**,** k**;**  cin **>>** u **>>** v **>>** k**;**  **--**u**;**  **--**v**;**  **if** **(**k**)**  ts**.**add**(**u**,** 0**,** v**,** 0**),** ts**.**add**(**u**,** 1**,** v**,** 1**),** ts**.**add**(**v**,** 0**,** u**,** 0**),** ts**.**add**(**v**,** 1**,** u**,** 1**);**  **else**  ts**.**add**(**u**,** 0**,** v**,** 1**),** ts**.**add**(**u**,** 1**,** v**,** 0**),** ts**.**add**(**v**,** 0**,** u**,** 1**),** ts**.**add**(**v**,** 1**,** u**,** 0**);**  **}**  int k **=** ts**.**ok**();**  **if** **(!**k**)**  cout **<<** "Impossible\n"**;**  **else**  **{**  vector**<**int**>** v**;**  **for** **(**int i **=** 0**;** i **<** n**;** i**++)**  **if** **(**ts**.**res**[**i**])**  v**.**push\_back**(**i**);**  cout **<<** **(**int**)**v**.**size**()** **<<** '\n'**;**  **for** **(**auto x **:** v**)**  cout **<<** x **+** 1 **<<** ' '**;**  cout **<<** '\n'**;**  **}**  **return** 0**;**  **}** |
|  |
| |  | | --- | | * *Check đồ thị hai phía* |   vector **<**int**>** adj**[**1000**];**  vector **<**int**>** side**(**1000**,** **-**1**);**  bool is\_bipartite **=** **true;**    void check\_bipartite**(**int u**)**  **{**  **for** **(**int i **=** 0**;** i **<** adj**[**u**].**size**();** i**++)**  **{**  int v **=** adj**[**u**][**i**];**  **if** **(**side**[**v**]** **==** **-**1**)**  **{**  side**[**v**]** **=** 1 **-** side**[**u**];**  check\_bipartite**(**v**);**  **}**  **else** **if** **(**side**[**u**]** **==** side**[**v**])**  is\_bipartite **=** **false;**  **}**  **}**    **for** **(**int u **=** 0**;** u **<** n**;** u**++)**  **{**  **if** **(**side**[**u**]** **==** **-**1**)**  **{**  side**[**u**]** **=** 0**;**  check\_bipartite**(**u**);**  **}**  **}** |
| |  | | --- | | * *Topo sort* |   void topo\_sort**(){**  **for(**int i **=** 1**;** i **<=** n**;** i**++)**  **{**  **if(**deg**[**i**]** **==** 0**)**  **{**  Q**.**push**(**i**);**  **}**  **}**  **while** **(!**Q**.**empty**()){**  int u **=** Q**.**front**();**  Q**.**pop**();**  x**[++**Size**]** **=** u**;**  **for** **(**int v **:** a**[**u**]){**  **if** **(--**deg**[**v**]** **==** 0**)** Q**.**push**(**v**);**  **}**  **}**  **}** |
| |  | | --- | | * *DSU* |   int FindSet**(**int x**)**  **{**  **if(** parent**[**x**]** **==** x**)** **return** x**;**  parent**[**x**]** **=** FindSet**(**parent**[**x**]** **);**  **return** FindSet**(**parent**[**x**]);**  **}**    void Union**(**int u**,** int v**)**  **{**  int parentU **=** FindSet**(**u**);**  int parentV **=** FindSet**(**v**);**  **if(** parentU **==** parentV**)**  **{**  **return;**  **}**  **if(**parentU **>** parentV**)**  parent**[**parentU**]** **=** parentV**;**  **else**  parent**[**parentV**]** **=** parentU**;**  **}** |
| |  | | --- | | * *LCA* |   int n**,** m**,** lg**;**  v List**[**N**];**  int deep**[**N**],** dad**[**N**][**20**];**  void DFS**(**int parent**,** int u**)**  **{**  deep**[**u**]** **=** deep**[**parent**]** **+** 1**;**  dad**[**u**][**0**]** **=** parent**;**  **for** **(**auto x **:** List**[**u**])**  **{**  **if** **(**x **==** parent**)**  **continue;**  DFS**(**u**,** x**);**  **}**  **}**  int LCA**(**int x**,** int y**)**  **{**  **if** **(**deep**[**x**]** **>** deep**[**y**])**  swap**(**x**,** y**);**  **for** **(**int i **=** 18**;** i **>=** 0**;** i**--)**  **{**  **if** **(**deep**[**y**]** **-** deep**[**x**]** **>=** **(**1 **<<** i**))**  y **=** dad**[**y**][**i**];**  **}**  **for** **(**int i **=** 18**;** i **>=** 0**;** i**--)**  **{**  **if** **(**dad**[**x**][**i**]** **!=** dad**[**y**][**i**])**  **{**  x **=** dad**[**x**][**i**];**  y **=** dad**[**y**][**i**];**  **}**  **}**  **if** **(**x **!=** y**)**  x **=** dad**[**x**][**0**];**  **return** x**;**  **}**  void solution**()**  **{**  cin **>>** n**;**  **for** **(**int i **=** 1**;** i **<** n**;** i**++)**  **{**  int x**,** y**;**  cin **>>** x **>>** y**;**  List**[**x**].**push\_back**(**y**);**  List**[**y**].**push\_back**(**x**);**  **}**  DFS**(**0**,** 1**);**  **for** **(**int i **=** 1**;** i **<=** 18**;** i**++)**  **{**  **for** **(**int j **=** 1**;** j **<=** n**;** j**++)**  **{**  dad**[**j**][**i**]** **=** dad**[**dad**[**j**][**i **-** 1**]][**i **-** 1**];**  **}**  **}**  cin **>>** m**;**  int root **=** 1**;**  **while** **(**m**--)**  **{**  int x**,** y**;**  cin **>>** x **>>** y**;**  int tu **=** LCA**(**root**,** x**);**  int tv **=** LCA**(**root**,** y**);**  int res **=** **(**deep**[**tu**]** **>=** deep**[**tv**]** **?** tu **:** tv**);**  int tuv **=** LCA**(**x**,** y**);**  res **=** **(**deep**[**res**]** **>=** deep**[**tuv**]** **?** res **:** tuv**);**  cout **<<** res **<<** endl**;**  **}**  **}** |
| |  | | --- | | * *Euler Tour* |   struct LCA  **{**  v height**,** euler**,** first**,** seg**,** vis**;**  int m**;**  void init**(**int root **=** 1**)**  **{**  height**.**resize**(**n **+** 5**);**  first**.**resize**(**n **+** 5**);**  vis**.**resize**(**n **+** 5**,** 0**);**  dfs**(**root**);**  m **=** euler**.**size**();**  seg**.**resize**(**m **\*** 4**);**  build**(**1**,** 0**,** m **-** 1**);**  **}**  void dfs**(**int node**,** int h **=** 0**)**  **{**  vis**[**node**]** **=** 1**;**  height**[**node**]** **=** h**;**  first**[**node**]** **=** euler**.**size**();**  euler**.**pb**(**node**);**  **for** **(**auto x **:** g**[**node**])**  **{**  **if** **(**vis**[**x**])**  **continue;**  dfs**(**x**,** h **+** 1**);**  euler**.**pb**(**node**);**  **}**  **}**  void build**(**int node**,** int l**,** int r**)**  **{**  **if** **(**l **==** r**)**  **{**  seg**[**node**]** **=** euler**[**l**];**  **}**  **else**  **{**  int m **=** **(**l **+** r**)** **/** 2**;**  build**(**node **\*** 2**,** l**,** m**);**  build**(**node **\*** 2 **+** 1**,** m **+** 1**,** r**);**  int x **=** seg**[**node **\*** 2**],** y **=** seg**[**node **\*** 2 **+** 1**];**  seg**[**node**]** **=** **(**height**[**x**]** **<** height**[**y**])** **?** x **:** y**;**  **}**  **}**  int get**(**int node**,** int L**,** int R**,** int l**,** int r**)**  **{**  **if** **(**L **>** r **||** l **>** R**)**  **return** **-**1**;**  **if** **(**l **<=** L **&&** R **<=** r**)**  **return** seg**[**node**];**  int m **=** **(**L **+** R**)** **/** 2**;**  int x **=** get**(**node **\*** 2**,** L**,** m**,** l**,** r**);**  int y **=** get**(**node **\*** 2 **+** 1**,** m **+** 1**,** R**,** l**,** r**);**  **if** **(**x **==** **-**1**)**  **return** y**;**  **if** **(**y **==** **-**1**)**  **return** x**;**  **return** height**[**x**]** **<** height**[**y**]** **?** x **:** y**;**  **}**  int lca**(**int x**,** int y**)**  **{**  int l **=** first**[**x**],** r **=** first**[**y**];**  **if** **(**l **>** r**)**  swap**(**l**,** r**);**  **return** get**(**1**,** 0**,** m **-** 1**,** l**,** r**);**  **}**  **};** |
| |  | | --- | | * *Heavy Light* |   // Truy vấn thay đổi trọng số trên cây  int n**;**  vii g**[**N**],** lst**;**  int dad**[**N**],** heavy**[**N**],** sub**[**N**],** cost**[**N**],** deep**[**N**],** pos**[**N**],** chain**[**N**],** head**[**N**];**  int seg**[**N **\*** 4**];**  void update**(**int node**,** int l**,** int r**,** int x**,** int val**)**  **{**  **if** **(**l **==** r**)**  **{**  seg**[**node**]** **=** val**;**  **return;**  **}**  int m **=** **(**l **+** r**)** **/** 2**;**  **if** **(**x **<=** m**)**  update**(**node **\*** 2**,** l**,** m**,** x**,** val**);**  **else**  update**(**node **\*** 2 **+** 1**,** m **+** 1**,** r**,** x**,** val**);**  seg**[**node**]** **=** max**(**seg**[**node **\*** 2**],** seg**[**node **\*** 2 **+** 1**]);**  **}**  int get**(**int node**,** int l**,** int r**,** int x**,** int y**)**  **{**  **if** **(**x **<=** l **&&** r **<=** y**)**  **{**  **return** seg**[**node**];**  **}**  **if** **(**l **>** y **||** r **<** x**)**  **return** 0**;**  int m **=** **(**l **+** r**)** **/** 2**;**  **return** max**(**get**(**node **\*** 2**,** l**,** m**,** x**,** y**),** get**(**node **\*** 2 **+** 1**,** m **+** 1**,** r**,** x**,** y**));**  **}**  void dfs**(**int node**,** int pre**)**  **{**  dad**[**node**]** **=** pre**;**  heavy**[**node**]** **=** **-**1**;**  sub**[**node**]** **=** 1**;**  **for** **(**auto x **:** g**[**node**])**  **{**  **if** **(**x**.**fi **==** pre**)**  **continue;**  cost**[**x**.**fi**]** **=** x**.**se**;**  deep**[**x**.**fi**]** **=** deep**[**node**]** **+** 1**;**  dfs**(**x**.**fi**,** node**);**  sub**[**node**]** **+=** sub**[**x**.**fi**];**  **if** **(**heavy**[**node**]** **==** **-**1 **||** sub**[**heavy**[**node**]]** **<** sub**[**x**.**fi**])**  **{**  heavy**[**node**]** **=** x**.**fi**;**  **}**  **}**  **}**  void init**()**  **{**  dfs**(**1**,** 1**);**  int num **=** 0**,** position **=** 1**;**  For**(**i**,** 1**,** n**)**  **{**  **if** **(**dad**[**i**]** **==** **-**1 **||** heavy**[**dad**[**i**]]** **!=** i**)**  **{**  num**++;**  **for** **(**int k **=** i**;** k **!=** **-**1**;** k **=** heavy**[**k**])**  **{**  pos**[**k**]** **=** position**++;**  update**(**1**,** 1**,** n**,** pos**[**k**],** cost**[**k**]);**  chain**[**k**]** **=** num**;**  head**[**k**]** **=** i**;**  **}**  **}**  **}**  **}**  int query**(**int x**,** int y**)**  **{**  int ans **=** 0**;**  **while** **(**chain**[**x**]** **!=** chain**[**y**])**  **{**  **if** **(**deep**[**head**[**x**]]** **>** deep**[**head**[**y**]])**  swap**(**x**,** y**);**  ans **=** max**(**ans**,** get**(**1**,** 1**,** n**,** pos**[**head**[**y**]],** pos**[**y**]));**  y **=** dad**[**head**[**y**]];**  **}**  **if** **(**deep**[**x**]** **>** deep**[**y**])**  swap**(**x**,** y**);**  **if** **(**deep**[**head**[**x**]]** **<=** deep**[**y**])**  **{**  ans **=** max**(**ans**,** get**(**1**,** 1**,** n**,** pos**[**heavy**[**x**]],** pos**[**y**]));**  **}**  **return** ans**;**  **}**  void solution**()**  **{**  cin **>>** n**;**  lst**.**pb**({-**1**,** **-**1**});**  For**(**i**,** 1**,** n **-** 1**)**  **{**  int x**,** y**,** w**;**  cin **>>** x **>>** y **>>** w**;**  g**[**x**].**pb**({**y**,** w**});**  g**[**y**].**pb**({**x**,** w**});**  lst**.**pb**({**x**,** y**});**  **}**  init**();**  For**(**i**,** 1**,** n **-** 1**)**  **{**  **if** **(**dad**[**lst**[**i**].**fi**]** **==** lst**[**i**].**se**)**  swap**(**lst**[**i**].**fi**,** lst**[**i**].**se**);**  **}**  int q**;**  cin **>>** q**;**  For**(**\_q**,** 1**,** q**)**  **{**  string type**;**  cin **>>** type**;**  **if** **(**type **==** "QUERY"**)**  **{**  int x**,** y**;**  cin **>>** x **>>** y**;**  cout **<<** query**(**x**,** y**)** **<<** endl**;**  **}**  **else**  **{**  int i**,** val**;**  cin **>>** i **>>** val**;**  update**(**1**,** 1**,** n**,** pos**[**lst**[**i**].**se**],** val**);**  **}**  **}**  **}** |
| |  | | --- | | * *Tarjan – TPLT mạnh* |   const int maxn **=** 1e6 **+** 1**;**  const int available **=** **-**1**;**  const int deleted **=** **-**2**;**  int n**,** m**;**  vector**<**int**>** Adj**[**maxn**];**  int num**[**maxn**],** low**[**maxn**];**  stack**<**int**>** Stack**;**    void Input**()**  **{**  cin **>>** n **>>** m**;**  **for** **(**int i **=** 1**;** i **<=** m**;** i**++)**  **{**  int u**,** v**;**  cin **>>** u **>>** v**;**  Adj**[**u**].**push\_back**(**v**);**  **}**  **}**    inline void Minimize**(**int**&** Target**,** int Value**)**  **{**  **if** **(**Value **<** Target**)**  Target **=** Value**;**  **}**    void DFSVisit**(**int u**)**  **{**  static int Time **=** 0**;**  num**[**u**]** **=** Time**++;**  low**[**u**]** **=** maxn**;**  Stack**.**push**(**u**);**    **for** **(**int v**:** Adj**[**u**])**  **{**  **if** **(**num**[**v**]** **==** deleted**)** **continue;**  **if** **(**num**[**v**]** **!=** available**)**  Minimize**(**low**[**u**],** num**[**v**]);**  **else**  **{**  DFSVisit**(**v**);**  Minimize**(**low**[**u**],** low**[**v**]);**  **}**  **}**    **if** **(**low**[**u**]** **>=** num**[**u**])**  **{**  int v**;**  **do**  **{**  v **=** Stack**.**top**();** Stack**.**pop**();**  num**[**v**]** **=** deleted**;**  **}**  **while** **(**v **!=** u**);**  **}**  **}**    void Tarjan**()**  **{**  fill**(**num **+** 1**,** num **+** 1 **+** n**,** available**);**  **for** **(**int u **=** 1**;** u **<=** n**;** u**++)**  **if** **(**num**[**u**]** **==** **-**1**)**  DFSVisit**(**u**);**  **}** |
| |  | | --- | | * *Khớp, cầu* |   const int maxN **=** 1e5 **+** 5**;**  int n**,** m**,** num**[**maxN**],** low**[**maxN**],** visit**[**maxN**],** tail**[**maxN**],** parent**[**maxN**],** timeDfs **=** 0**;**  bool root**[**maxN**];**  vector**<**vector**<**int**>>** adj**(**maxN**);**  void dfs**(**int u**,** int prev**)**  **{**  visit**[**u**]** **=** 1**;**  parent**[**u**]** **=** prev**;**  timeDfs**++;**  num**[**u**]** **=** timeDfs**;**  low**[**u**]** **=** timeDfs**;**  **for** **(**auto v **:** adj**[**u**])**  **{**  **if** **(**v **!=** prev**)**  **{**  **if** **(**visit**[**v**]** **==** 0**)**  **{**  dfs**(**v**,** u**);**  low**[**u**]** **=** min**(**low**[**u**],** low**[**v**]);**  **}**  **else**  low**[**u**]** **=** min**(**low**[**u**],** num**[**v**]);**  **}**  **}**  tail**[**u**]** **=** timeDfs**;**  **}**  int findKhop**()**  **{**  int Count **=** 0**;**  **for** **(**int u **=** 1**;** u **<=** n**;** u**++)**  **{**  **if** **(**root**[**u**]** **&&** adj**[**u**].**size**()** **<=** 1**)**  **continue;**  **for** **(**auto v **:** adj**[**u**])**  **{**  **if** **(**root**[**u**])**  **{**  **if** **(**u **==** parent**[**v**]** **&&** low**[**v**]** **>** num**[**u**])**  **{**  Count**++;**  **break;**  **}**  **}**  **else** **if** **(**u **==** parent**[**v**]** **&&** low**[**v**]** **>=** num**[**u**])**  **{**  Count**++;**  // cout << u << " " << v << '\n';  **break;**  **}**  **}**  **}**  **return** Count**;**  **}**  int findCau**()**  **{**  int Count **=** 0**;**  **for** **(**int u **=** 1**;** u **<=** n**;** u**++)**  **{**  **for** **(**auto v **:** adj**[**u**])**  **{**  **if** **(**u **==** parent**[**v**]** **&&** low**[**v**]** **>** num**[**u**])**  **{**  Count**++;**  **}**  **}**  **}**  **return** Count**;**  **}**  void solve**()**  **{**  cin **>>** n **>>** m**;**  FOR**(**i**,** 1**,** n**)**  root**[**i**]** **=** **false;**  int u**,** v**;**  FOR**(**i**,** 1**,** m**)**  **{**  cin **>>** u **>>** v**;**  adj**[**u**].**push\_back**(**v**);**  adj**[**v**].**push\_back**(**u**);**  **}**  FOR**(**i**,** 1**,** n**)**  **{**  **if** **(**visit**[**i**]** **==** 0**)**  **{**  root**[**i**]** **=** **true;**  dfs**(**i**,** 0**);**  **}**  **}**  cout **<<** findKhop**()** **<<** " " **<<** findCau**()** **<<** endl**;**  **}** |
| |  | | --- | | * *Dijkstra on Segment Tree* |     #include<bits/stdc++.h>  **using** **namespace** std**;**  const int N **=** 1e5 **+** 9**;**  vector**<**pair**<**int**,** int**>>** g**[**N **\*** 9**];**  inline void add\_edge**(**int u**,** int v**,** int w**)** **{**  g**[**u**].**push\_back**({**v**,** w**});**  **}**  int add**;**  void build**(**int n**,** int b**,** int e**)** **{**  **if** **(**b **==** e**)** **{**  add\_edge**(**b**,** n **+** add**,** 0**);**  add\_edge**(**n **+** add **\*** 5**,** b**,** 0**);**  **return;**  **}**  int mid **=** b **+** e **>>** 1**;**  add\_edge**(**2 **\*** n **+** add**,** n **+** add**,** 0**);**  add\_edge**(**2 **\*** n **+** 1 **+** add**,** n **+** add**,** 0**);**  add\_edge**(**n **+** 5 **\*** add**,** 2 **\*** n **+** 5 **\*** add**,** 0**);**  add\_edge**(**n **+** 5 **\*** add**,** 2 **\*** n **+** 1 **+** 5 **\*** add**,** 0**);**  build**(**2 **\*** n**,** b**,** mid**);**  build**(**2 **\*** n **+** 1**,** mid **+** 1**,** e**);**  **}**  void upd**(**int n**,** int b**,** int e**,** int i**,** int j**,** int dir**,** int u**,** int w**)** **{**  **if** **(**j **<** b **||** e **<** i**)** **return;**  **if** **(**i **<=** b **&&** e **<=** j**)** **{**  **if** **(**dir**)** add\_edge**(**u**,** n **+** 5 **\*** add**,** w**);** // from u to this range  **else** add\_edge**(**n **+** add**,** u**,** w**);** // from this range to u  **return;**  **}**  int mid **=** **(**b **+** e**)** **>>** 1**;**  upd**(**2 **\*** n**,** b**,** mid**,** i**,** j**,** dir**,** u**,** w**);**  upd**(**2 **\*** n **+** 1**,** mid **+** 1**,** e**,** i**,** j**,** dir**,** u**,** w**);**  **}**  vector**<**long long**>** dijkstra**(**int s**)** **{**  const long long inf **=** 1e18**;**  priority\_queue**<**pair**<**long long**,** int**>,** vector**<**pair**<**long long**,** int**>>,** greater**<**pair**<**long long**,** int**>>>** q**;**  vector**<**long long**>** d**(**9 **\*** N **+** 1**,** inf**);** vector**<**bool**>** vis**(**9 **\*** N **+** 1**,** 0**);**  q**.**push**({**0**,** s**});**  d**[**s**]** **=** 0**;**  **while(!**q**.**empty**()){**  auto x **=** q**.**top**();** q**.**pop**();**  int u **=** x**.**second**;**  **if(**vis**[**u**])** **continue;** vis**[**u**]** **=** 1**;**  **for(**auto y**:** g**[**u**]){**  int v **=** y**.**first**;** long long w **=** y**.**second**;**  **if(**d**[**u**]** **+** w **<** d**[**v**]){**  d**[**v**]** **=** d**[**u**]** **+** w**;** q**.**push**({**d**[**v**],** v**});**  **}**  **}**  **}**  **return** d**;**  **}**  long long ans**[**N**];**  int32\_t main**()** **{**  ios\_base**::**sync\_with\_stdio**(**0**);**  cin**.**tie**(**0**);**  int n**,** q**,** s**;** cin **>>** n **>>** q **>>** s**;**  add **=** n**;**  build**(**1**,** 1**,** n**);**  **while** **(**q**--)** **{**  int ty**;** cin **>>** ty**;**  int u**,** l**,** r**,** w**;**  **if** **(**ty **==** 1**)** **{**  cin **>>** u **>>** l **>>** w**;**  r **=** l**;**  **}**  **else** **{**  cin **>>** u **>>** l **>>** r **>>** w**;**  **}**  upd**(**1**,** 1**,** n**,** l**,** r**,** ty **<=** 2**,** u**,** w**);**  **}**  auto ans **=** dijkstra**(**s**);**  **for** **(**int i **=** 1**;** i **<=** n**;** i**++)** **{**  **if** **(**ans**[**i**]** **==** 1e18**)** ans**[**i**]** **=** **-**1**;**  cout **<<** ans**[**i**]** **<<** ' '**;**  **}**  **return** 0**;**  **}**  // https://codeforces.com/contest/786/problem/B |
| |  | | --- | | * *Cạnh min, max giữa hai đỉnh x, y bất kì trên cây* |   ll h**[**maxN**]** **=** **{};**  vector**<**ii**>** Adj**[**maxN**];**  ll p**[**maxN**][**20**]** **=** **{};**  struct data  **{**  int maxC **=** **-**INFINITY**,** minC **=** INFINITY**;**  **}**up**[**maxN**][**20**];**  void DFS**(**int u**)**  **{**  // cout << u << "\n";  **for** **(**auto e **:** Adj**[**u**])**  **{**  int v **=** e**.**first**;**  int c **=** e**.**second**;**  **if** **(**h**[**v**]** **==** **-**1**)**  **{**  p**[**v**][**0**]** **=** u**;**  h**[**v**]** **=** h**[**u**]** **+** 1**;**  DFS**(**v**);**  up**[**v**][**0**].**maxC **=** c**;**  up**[**v**][**0**].**minC **=** c**;**  **}**  **}**  **}**    void Prepare**()**  **{**  memset**(**h**,** **-**1**,** **sizeof(**h**));**  memset**(**p**,** **-**1**,** **sizeof(**p**));**  // FOR(i,1,n)  // {  // cout << h[i] << " ";  // }  h**[**1**]** **=** 0**;** DFS**(**1**);**    **for** **(**int j **=** 1 **;** **(**1 **<<** j**)** **<=** n**;** j **++)**  **for** **(**int i **=** 1**;** i **<=** n**;** i**++)**  **if** **(**p**[**i**][**j **-** 1**]** **!=** **-**1**)**  **{**  p**[**i**][**j**]** **=** p**[**p**[**i**][**j **-** 1**]][**j **-** 1**];**  up**[**i**][**j**].**maxC **=** max**(**up**[**i**][**j **-** 1**].**maxC**,** up**[**p**[**i**][**j **-** 1**]][**j **-** 1**].**maxC**);**  up**[**i**][**j**].**minC **=** min**(**up**[**i**][**j **-** 1**].**minC**,** up**[**p**[**i**][**j **-** 1**]][**j **-** 1**].**minC**);**  **}**  **}**    data LCA**(**int u**,** int v**)**  **{**  data res**;**  **if** **(**h**[**u**]** **<** h**[**v**])** swap**(**u**,** v**);**  **for** **(**int i **=** 19**;** i **>=** 0**;** i**--)**  **if** **(**h**[**u**]** **-** **(**1 **<<** i**)** **>=** h**[**v**])**  **{**  res**.**maxC **=** max**(**res**.**maxC**,** up**[**u**][**i**].**maxC**);**  res**.**minC **=** min**(**res**.**minC**,** up**[**u**][**i**].**minC**);**  u **=** p**[**u**][**i**];**    **}**    **if** **(**u **==** v**)** **return** res**;**    **for** **(**int i **=** 19**;** i **>=** 0**;** i**--)**  **if** **(**p**[**u**][**i**]** **!=** p**[**v**][**i**])**  **{**  res**.**maxC **=** max**(**max**(**res**.**maxC**,** up**[**u**][**i**].**maxC**),** up**[**v**][**i**].**maxC**);**  res**.**minC **=** min**(**min**(**res**.**minC**,** up**[**u**][**i**].**minC**),** up**[**v**][**i**].**minC**);**  u **=** p**[**u**][**i**],** v **=** p**[**v**][**i**];**  **}**  res**.**maxC **=** max**(**max**(**res**.**maxC**,** up**[**u**][**0**].**maxC**),** up**[**v**][**0**].**maxC**);**  res**.**minC **=** min**(**min**(**res**.**minC**,** up**[**u**][**0**].**minC**),** up**[**v**][**0**].**minC**);**  **return** res**;**  **}** |
| |  | | --- | | * *Dinic* |   struct Edge  **{**  int u**,** v**;**  ll cap**,** flow**;**  Edge**()** **{}**  Edge**(**int \_u**,** int \_v**,** ll \_cap**)** **:** u**(**\_u**),** v**(**\_v**),** cap**(**\_cap**),** flow**(**0**)** **{}**  **};**  struct Dinic  **{**  int N**;**  vector**<**Edge**>** E**;**  vector**<**vector**<**int**>>** g**;**  vector**<**int**>** d**,** pt**;**  Dinic**(**int \_N**)** **:** N**(**\_N**),** E**(**0**),** g**(**\_N**),** d**(**\_N**),** pt**(**\_N**)** **{}**  void AddEdge**(**int u**,** int v**,** ll cap**)**  **{**  **if** **(**u **!=** v**)**  **{**  E**.**emplace\_back**(**Edge**(**u**,** v**,** cap**));**  g**[**u**].**emplace\_back**(**E**.**size**()** **-** 1**);**  E**.**emplace\_back**(**Edge**(**v**,** u**,** 0**));**  g**[**v**].**emplace\_back**(**E**.**size**()** **-** 1**);**  **}**  **}**  bool BFS**(**int S**,** int T**)**  **{**  queue**<**int**>** q**({**S**});**  fill**(**d**.**begin**(),** d**.**end**(),** N **+** 1**);**  d**[**S**]** **=** 0**;**  **while** **(!**q**.**empty**())**  **{**  int u **=** q**.**front**();**  q**.**pop**();**  **if** **(**u **==** T**)**  **break;**  **for** **(**int k **:** g**[**u**])**  **{**  Edge **&**e **=** E**[**k**];**  **if** **(**e**.**flow **<** e**.**cap **&&** d**[**e**.**v**]** **>** d**[**e**.**u**]** **+** 1**)**  **{**  d**[**e**.**v**]** **=** d**[**e**.**u**]** **+** 1**;**  q**.**emplace**(**e**.**v**);**  **}**  **}**  **}**  **return** d**[**T**]** **!=** N **+** 1**;**  **}**  ll DFS**(**int u**,** int T**,** ll flow **=** **-**1**)**  **{**  **if** **(**u **==** T **||** flow **==** 0**)**  **return** flow**;**  **for** **(**int **&**i **=** pt**[**u**];** i **<** g**[**u**].**size**();** **++**i**)**  **{**  Edge **&**e **=** E**[**g**[**u**][**i**]];**  Edge **&**oe **=** E**[**g**[**u**][**i**]** **^** 1**];**  **if** **(**d**[**e**.**v**]** **==** d**[**e**.**u**]** **+** 1**)**  **{**  ll amt **=** e**.**cap **-** e**.**flow**;**  **if** **(**flow **!=** **-**1 **&&** amt **>** flow**)**  amt **=** flow**;**  **if** **(**ll pushed **=** DFS**(**e**.**v**,** T**,** amt**))**  **{**  e**.**flow **+=** pushed**;**  oe**.**flow **-=** pushed**;**  **return** pushed**;**  **}**  **}**  **}**  **return** 0**;**  **}**  ll MaxFlow**(**int S**,** int T**)**  **{**  ll total **=** 0**;**  **while** **(**BFS**(**S**,** T**))**  **{**  fill**(**pt**.**begin**(),** pt**.**end**(),** 0**);**  **while** **(**ll flow **=** DFS**(**S**,** T**))**  total **+=** flow**;**  **}**  **return** total**;**  **}**  **};**  void init**()**  **{**  **}**  void solution**()**  **{**  ll n**,** m**,** s**,** t**;**  cin **>>** n **>>** m **>>** s **>>** t**;**  Dinic g**(**n **+** 5**);**  For**(**i**,** 1**,** m**){**  ll x**,** y**,** w**;**  cin **>>** x **>>** y **>>** w**;**  g**.**AddEdge**(**x**,** y**,** w**);**  **}**  cout **<<** g**.**MaxFlow**(**s**,** t**)** **<<** endl**;**  **}** |
| |  | | --- | | * *Fenwick Tree on Tree* |   #include <bits/stdc++.h>  #include <ext/pb\_ds/assoc\_container.hpp>  #include <ext/pb\_ds/tree\_policy.hpp>  #include <ext/pb\_ds/detail/standard\_policies.hpp>  #pragma GCC optimize("O3")  #pragma GCC target("sse4")  **using** **namespace** std**;**  **using** **namespace** \_\_gnu\_pbds**;**  **using** ll **=** long long**;**  **using** ull **=** unsigned long long**;**  **using** i128 **=** \_\_int128\_t**;**  **using** db **=** double**;**  **using** ii **=** pair**<**ll**,** ll**>;**  #define fi first  #define se second  #define in binary\_search  #define vec vector  #define pb push\_back  #define all(a) a.begin(), a.end()  #define umap unordered\_map  #define For(i, a, b) for (ll i = (ll)a; i <= (ll)b; i++)  #define Ford(i, a, b) for (ll i = (ll)a; i >= (ll)b; i--)  #define uniq(a) a.resize(unique(all(a)) - a.begin())  umap**<**ll**,** ll**>** compress**(**vec**<**ll**>** a**)**  **{**  sort**(**all**(**a**));**  uniq**(**a**);**  umap**<**ll**,** ll**>** ans**;**  For**(**i**,** 0**,** a**.**size**()** **-** 1**)** ans**[**a**[**i**]]** **=** i **+** 1**;**  **return** ans**;**  **}**  const ll N **=** 2e5 **+** 5**;**  ll n**;**  ll color**[**N**],** way**[**N**],** idx**[**N**],** s**[**N**],** vis**[**N**],** f**[**N**];**  vec**<**ll**>** g**[**N**];**  ll id **=** 1**;**  void update**(**ll i**,** ll x**)**  **{**  **for** **(;** i **<** N**;** i **+=** i **&** **-**i**)**  f**[**i**]** **+=** x**;**  **}**  ll query**(**ll i**)**  **{**  ll ans **=** 0**;**  **for** **(;** i**;** i **-=** i **&** **-**i**)**  ans **+=** f**[**i**];**  **return** ans**;**  **}**  void dfs**(**ll node**,** ll pre **=** 0**)**  **{**  // cout << node << endl;  way**[**node**]** **=** id**;**  idx**[**id**]** **=** node**;**  s**[**node**]** **=** 1**;**  id **+=** 1**;**  **for** **(**auto x **:** g**[**node**])**  **{**  **if** **(**x **==** pre**)**  **continue;**  dfs**(**x**,** node**);**  s**[**node**]** **+=** s**[**x**];**  **}**  **}**  void init**()**  **{**  **}**  void solution**()**  **{**  cin **>>** n**;**  For**(**i**,** 1**,** n **-** 1**)**  **{**  ll x**,** y**;**  cin **>>** x **>>** y**;**  g**[**x**].**pb**(**y**);**  g**[**y**].**pb**(**x**);**  **}**  // nén  vec**<**ll**>** c**;**  For**(**i**,** 1**,** n**)** cin **>>** color**[**i**],** c**.**pb**(**color**[**i**]);**  umap**<**ll**,** ll**>** pos **=** compress**(**c**);**  For**(**i**,** 1**,** n**)** color**[**i**]** **=** pos**[**color**[**i**]];**  //  dfs**(**1**);**  //  vec**<**tuple**<**ll**,** ll**,** ll**>>** ls**;**  For**(**i**,** 1**,** n**)** ls**.**pb**({**way**[**i**],** way**[**i**]** **+** s**[**i**]** **-** 1**,** i**});**  sort**(**all**(**ls**),** **[](**tuple**<**ll**,** ll**,** ll**>** x**,** tuple**<**ll**,** ll**,** ll**>** y**)**  **{** **return** get**<**1**>(**x**)** **>** get**<**1**>(**y**);** **});**  // For(i, 1, n) cout << way[i] << " ";  // cout << endl;  // For(i, 1, n) cout << idx[i] << " ";  // cout << endl;  // for(auto [l, r, x]: ls)  // cout << l << " " << r << " " << x << endl;  // return;  //  fill**(**vis**,** vis **+** n **+** 1**,** **-**1**);**  ll ans**[**n **+** 5**]** **=** **{**0**};**  For**(**i**,** 1**,** n**)**  **{**  ll x **=** color**[**idx**[**i**]];**  **if** **(**vis**[**x**]** **!=** **-**1**)**  update**(**vis**[**x**],** **-**1**);**  vis**[**x**]** **=** i**;**  update**(**i**,** 1**);**  **while** **(**ls**.**size**()** **&&** get**<**1**>(**ls**.**back**())** **==** i**)**  **{**  auto **[**l**,** r**,** x**]** **=** ls**.**back**();**  ans**[**x**]** **=** query**(**r**)** **-** query**(**l **-** 1**);**  ls**.**pop\_back**();**  **}**  **}**  For**(**i**,** 1**,** n**)** cout **<<** ans**[**i**]** **<<** endl**;**  **}** |
| |  | | --- | | * *DSU on Tree* |   #include <bits/stdc++.h>  #define sz(a) (int)a.size()  **using** **namespace** std**;**  const int mN **=** 1e5 **+** 10**;**  int numNode**;**  int cntArc**[**mN**],** toArc**[**mN**],** r**[**mN**],** res**[**mN**];**  queue **<**int**>** q**;**  vector **<**int**>** adj**[**mN**];**  set **<**int**>** colors**[**mN**];**  void dfs**(**int u**,** int pa**)** **{**  **for** **(**int v **:** adj**[**u**])** **if** **(**v **!=** pa**)** **{**  cntArc**[**u**]++;**  toArc**[**v**]** **=** u**;**  dfs**(**v**,** u**);**  **}**  **}**  int main**()** **{**  int u**,** v**,** c**,** ru**,** rv**;**  cin **>>** numNode**;**  **for** **(**int i **=** 1**;** i **<** numNode**;** i**++)** **{**  cin **>>** u **>>** v**;**  adj**[**u**].**push\_back**(**v**);**  adj**[**v**].**push\_back**(**u**);**  **}**  dfs**(**1**,** 1**);**  **for** **(**int i **=** 1**;** i **<=** numNode**;** i**++)** **{**  cin **>>** c**;**  **if** **(**sz**(**adj**[**i**])** **==** 1**)** q**.**push**(**i**);**  colors**[**i**].**insert**(**c**);**  r**[**i**]** **=** i**;**  **}**  **while** **(**sz**(**q**))** **{**  u **=** q**.**front**();**  q**.**pop**();**  v **=** toArc**[**u**];**  ru **=** r**[**u**];**  rv **=** r**[**v**];**  res**[**u**]** **=** sz**(**colors**[**ru**]);**  **if** **(**sz**(**colors**[**ru**])** **<** sz**(**colors**[**rv**]))** swap**(**ru**,** rv**);**  **for** **(**int color **:** colors**[**rv**])** colors**[**ru**].**insert**(**color**);**  colors**[**v**].**clear**();**  r**[**v**]** **=** ru**;**  cntArc**[**v**]--;**  **if** **(!**cntArc**[**v**])** q**.**push**(**v**);**  **}**  **for** **(**int i **=** 1**;** i **<=** numNode**;** i**++)** cout **<<** res**[**i**]** **<<** "\n"**;**  **}** |

# **6. String**

|  |  |
| --- | --- |
| |  | | --- | | * *Hashing use with sum prefix* |   const int maxN**=** 1e6 **+** 5**;**  int n**;**  string T**,**P**;**  ll hashT**[**maxN**];**  const int base **=** 1e9 **+** 7**;**  ll Pow**[**maxN**]** **=** **{};**  ll getHashT**(**int n**,** int m**)**  **{**  ll pre **=** hashT**[**n**-**1**]** **\*** Pow**[**m **-** n **+** 1**]** **%** base**;**  **if(** pre **>** hashT**[**m**])** hashT**[**m**]** **+=** base**;**  **return** **(**hashT**[**m**]** **-** pre**)** **%** base**;**  **}**  ll hashP **=** 0**;**  void prepare**()**  **{**  hashP **=** 0**;**  FOR**(**i**,**1**,**P**.**length**())**  **{**  hashP **=** **(**hashP **\*** 26 **+** P**[**i**-**1**]** **-** 'a'**)** **%** base**;**  **}**  hashT**[**0**]** **=** 0**;**  FOR**(**i**,**1**,**T**.**length**())**  **{**  hashT**[**i**]** **=** **(**hashT**[**i**-**1**]** **\*** 26 **+** T**[**i**-**1**]** **-** 'a'**)** **%** base**;**  **}**  **}** |
| |  | | --- | | * *Hasing use with segment tree* |   const int maxN**=** 1e5 **+** 3**;**  const int base **=** 1e9 **+** 7**;**  const int base2 **=** 1e9 **+** 9**;**  struct node  **{**  int length**;**  ll hashValue**;**  ll hashValue2**;**  **};**  ll Pow**[**maxN**]** **=** **{};**  ll sum**[**maxN**]** **=** **{};**  ll Pow2**[**maxN**]** **=** **{};**  ll sum2**[**maxN**]** **=** **{};**  node ST**[**4 **\*** maxN**]** **=** **{};**  int lazy**[**4 **\*** maxN**]** **=** **{};**    node Merge**(**node a**,** node b**)**  **{**  node ans**;**  ans**.**hashValue **=** 0**;**  ans**.**hashValue2 **=** 0**;**  ans**.**length **=** a**.**length **+** b**.**length**;**  ans**.**hashValue **=** **(**ans**.**hashValue **+** a**.**hashValue **\*** Pow**[**b**.**length**]** **%** base **+** b**.**hashValue**)** **%** base**;**  ans**.**hashValue2 **=** **(**ans**.**hashValue2 **+** a**.**hashValue2 **\*** Pow2**[**b**.**length**]** **%** base2 **+** b**.**hashValue2**)** **%** base2**;**  **return** ans**;**  **}**  //hashing part  string T**,**P**;**    void prepare**()**  **{**  memset**(**lazy**,** **-**1**,** **sizeof(**lazy**));**  Pow**[**0**]** **=** 1**;**  Pow2**[**0**]** **=** 1**;**  Pow**[**1**]** **=** 10**;**  Pow2**[**1**]** **=** 10**;**  sum**[**1**]** **=** 1**;**  sum2**[**1**]** **=** 1**;**  FOR**(**i**,**2**,**maxN**)**  **{**  Pow**[**i**]** **=** **(**Pow**[**i**-**1**]** **\*** 10**)** **%** base**;**  Pow2**[**i**]** **=** **(**Pow2**[**i**-**1**]** **\*** 10**)** **%** base2**;**  sum**[**i**]** **=** **(**sum**[**i**-**1**]** **+** Pow**[**i **-** 1**])** **%** base**;**  sum2**[**i**]** **=** **(**sum2**[**i**-**1**]** **+** Pow2**[**i**-**1**])** **%** base2**;**  **}**  **}**  void build**(**int id**,** int l**,** int r**)**  **{**  **if(** l **==** r**)**  **{**    **return;**  **}**  int mid **=** **(**l **+** r**)** **/** 2**;**  build**(**id **\*** 2**,** l**,** mid**);**  build**(**id **\*** 2 **+** 1**,** mid **+** 1**,** r**);**  ST**[**id**]** **=** Merge**(**ST**[**id **\*** 2**],** ST**[**id **\*** 2 **+** 1**]);**  **}**      void down**(**int id**)**  **{**  //11.1111  //  ST**[**id **\*** 2**].**hashValue **=** sum**[**ST**[**id **\*** 2**].**length**]** **\*** lazy**[**id**]** **%** base**;**  ST**[**id **\*** 2**].**hashValue2 **=** sum2**[**ST**[**id **\*** 2**].**length**]** **\*** lazy**[**id**]** **%** base2**;**    ST**[**id **\*** 2 **+** 1**].**hashValue **=** sum**[**ST**[**id **\*** 2 **+** 1**].**length**]** **\*** lazy**[**id**]** **%** base**;**  ST**[**id **\*** 2 **+** 1**].**hashValue2 **=** sum2**[**ST**[**id **\*** 2 **+** 1**].**length**]** **\*** lazy**[**id**]** **%** base2**;**    lazy**[**id **\*** 2**]** **=** lazy**[**id**];**  lazy**[**id **\*** 2 **+** 1**]** **=** lazy**[**id**];**    lazy**[**id**]** **=** **-**1**;**  **}**    void update**(**int id**,** int l**,** int r**,** int u**,** int v**,** ll value**)**  **{**  //cout << id << " " << l << " " << r << " " << value << '\n';  **if(** v **<** l **||** r **<** u**)**  **{**  **return;**  **}**  **if(** u **<=** l **&&** r **<=**v **)**  **{**  lazy**[**id**]** **=** value**;**  ST**[**id**].**hashValue **=** value **\*** sum**[**ST**[**id**].**length**]** **%** base**;**  ST**[**id**].**hashValue2 **=** value **\*** sum2**[**ST**[**id**].**length**]** **%** base2**;**  **return;**  **}**  int mid **=** **(**l **+** r**)** **/** 2**;**  //day xuong  **if(**lazy**[**id**]** **!=** **-**1**)**  down**(**id**);**  //1..45....9  update**(**id **\*** 2**,** l**,** mid**,** u**,** v**,** value**);**  update**(**id **\*** 2 **+** 1**,** mid **+** 1**,** r**,** u**,** v**,** value**);**  ST**[**id**]** **=** Merge**(**ST**[**id **\*** 2**],** ST**[**id **\*** 2 **+** 1**]);**  **}**    node get**(**int id**,** int l**,** int r**,** int u**,** int v**)**  **{**  // cout << id << " " << l << " " << r << '\n';  **if(** v **<** l **||** r **<** u**)**  **{**  node ans **=** **{**0**,** 0**,** 0**};**  **return** ans**;**  **}**  **if(** u **<=** l **&&** r **<=** v**)**  **{**  **return** ST**[**id**];**  **}**  int mid **=** **(**l **+** r**)** **/** 2**;**  //day xuong  **if(**lazy**[**id**]** **!=** **-**1**)**  down**(**id**);**    node f **=** get**(**id **\*** 2**,** l**,** mid**,** u**,** v**);**  node s **=** get**(**id **\*** 2 **+** 1**,** mid **+** 1**,** r**,** u**,** v**);**  **return** Merge**(**f**,** s**);**  **}** |
| |  | | --- | | * *KMP* |   int lps**[**maxN**]** **=** **{};**  int n**,**m**;**  void kmpPreprocess**(**string s**,** int n**)**  **{**  int j **=** 1**;**  lps**[**1**]** **=** 0**;**  FOR**(**i**,**2**,**n**)**  **{**  j **=** lps**[**i**-**1**];**  **while(**j **>** 0 **&&** s**[**j**+**1**]** **!=** s**[**i**])**  **{**  j **=** lps**[**j**];**  **}**  **if(**s**[**j**+**1**]** **==** s**[**i**])**  **{**  lps**[**i**]** **=** j **+** 1**;**  **}**  **else** lps**[**i**]** **=** 0**;**  **}**  **}**  void kmpSearch**()**  **{**  //lps is set up for string p (string you have to find on string s)    int j **=** 0**;**  FOR**(**i**,**1**,**n**)**  **{**  **while(**p**[**j**+**1**]** **!=** s**[**i**]** **&&** j **>** 0**)**  **{**  j **=** lps**[**j**];**  **}**  **if(**p**[**j**+**1**]** **==** s**[**i**])** j**++;**  **if(**j **==** m**)**  **{**  cout **<<** i **-** j **+** 1 **<<** " "**;**  j **=** lps**[**j**];**  **}**  **}**  **}** |