Cleber-cvsj - ICPC Library

8 de setembro de 2025

Conteúdo

1 graphs

1.1 Diksktra

```
CF0 const ll MAXN = 1e5+5;
B6E const ll INF = LLONG_MAX;

F79 #define pll pair<long long, long long>
0C1 #define vi vector<int>
358 #define vll vector<long long>

EB6 vector<ll> dist(MXN,INF);
9BC vector<pii> graph[MAXN];

C6B void dikstra(int x,int n,vector<bool>vis){

336     dist[x] = 0;
F37     priority_queue<pii, vector<pii>, greater<pii>> pq;
E63     pq.push({0,x}); //uma especia de bfs so que com pq
502     while(!pq.empty()){
A5B         int a = pq.top().second;
A3C         int b = pq.top().first;
716         pq.pop();
9F6         if(b >dist[out] ) continue;
721         for([cost, nb] : graph[a]){
C1A             ll currD = dist[out] + cost;
BE5             if(currD < dist[nb]){
2FF               dist[nb] = currD;
F64               parent[nb] = out;
A33               pq.push({currD,nb})
CD7             }
0EA         }
2FA     }
E1C }
```

1.2 MST

```
2B7 #include<bits/stdc++.h>

F79 #define pll pair<long long, long long>
0C1 #define vi vector<int>
358 #define vll vector<long long>

AD1 typedef long long ll;


4C9 vi liga;
607 vi tam;

57B int find(int x){
924   if(x == liga[x]){
EA5     return x;
2EF   }
AB5   return liga[x] = find(liga[x]);
D68 }


3D2 bool same(int a,int b){
C0A     return find(a) == find(b);
B5D }

440 void unite(int a, int b){
BCA     a = find(a);
B88     b = find(b);
226     if (tam[a] < tam[b]) swap(a,b);
AD6     tam[a]+= tam[b];
10D     liga[b]=a;
F33 }



E8D int main(){
96E     ios::sync_with_stdio(NULL);
B95     cin.tie(0);
221     int t; cin >> t;

EE2     while(t--){
8AD         int p,n,m;cin >> p >> n >> m;
53A         int peso = 0;
FB0         liga.resize(n+1);
DE6         tam.resize(n+1);
78A         for (int i = 1; i <= n; i++){
CD6             liga[i] = i;
8A8             tam[i] = 1;
EFB         }
508         vector<pair<int,pii>>arestas;
DD5         for(int i = 0; i < m; i++){
684             int a,b,w;
060             cin >> a >> b >>w;
C65             arestas.push_back(make_pair(w,make_pair(a,b)))
    ;
7DE         }
DA0         sort(arestas.begin(),arestas.end());

DD5         for(int i = 0; i < m; i++){
6DE             if(!same(arestas[i].second.first,arestas[i].
    second.second)){
854                 unite(arestas[i].second.first,arestas[i].
    second.second);
E0E                 peso+= arestas[i].first;
3B9             }
78B         }
74C         cout << p * peso << '\n';

696     }


BB3     return 0;
C35 }
```

1.3 DijkstraGrid

```
EF9 const int ms = 501;
F50 bool vis[ms][ms];
14E int n,m;
A75 int grid[ms][ms];
7CC int dist[ms][ms];

89C bool inRange(int x, int y){
8E7   return x>=0 && x < n && y>=0 && y <m;
6C4 }
495 int dx[] = {0,0,1,-1};
B0A int dy[] = {1,-1,0,0};
D53 int bfs(){
8C1   priority_queue<pair<int,pii>,vector<pair<int,pii>>,
  greater<pair<int,pii>>>pq;
606   memset(vis,false,sizeof(vis));
6AD   for(int i = 0; i < ms; i++) {
850     for(int j = 0; j < ms; j++) {
AE8       dist[i][j] = 1000000;
39D     }
D66 }

72E   vis[0][0] = 1;
8E7   dist[0][0]= 0;
C8C   pq.push({0,{0,0}});
669   int d = 0;
502   while(!pq.empty()){
557     int sz = pq.size();
2F9     while(sz--){
52A       int d =pq.top().first;
9F3       int x = pq.top().second.first;
B8D       int y = pq.top().second.second;
716       pq.pop();
11E       if(x == n-1 && y == m-1) return dist[n-1][m-1];
1CD       for(int i = 0; i < 4; i++){
```

```
6EB         int ax = x + dx[i], ay = y +dy[i];
A3A         if(inRange(ax,ay) && !vis[ax][ay]){
078           int camim = 1-grid[ax][ay];
2C6           if(dist[x][y] + camim < dist[ax][ay]){
8CE             vis[ax][ay] =1;
F97             dist[ax][ay] = dist[x][y] + camim;
EC5             pq.push({dist[ax][ay],{ax,ay}});
57D           }
0E0         }
B45       }
12E     }
D74   }
7F7 }
```

1.4 toposort

```
F79 #define pll pair<long long, long long>
0C1 #define vi vector<int>
358 #define vll vector<long long>

468 #define ms 10001

0C5 void bfs(vector<vi>&graph,vector<int>indegree,int n,vector
    <int>&topo){
451   priority_queue<int, vector<int>,greater<int>>PQ;

78A   for (int i = 1; i <= n; i++){
D9F     if(indegree[i] == 0){
704       PQ.push(i);
784     }
14F   }

849   while(!PQ.empty()){
16E     int u = PQ.top();
41B     PQ.pop();
AB4     topo.push_back(u);
577     for (int v : graph[u]){
35C       if(--indegree[v] == 0){
27C         PQ.push(v);
01A       }
EA7     }
2B1   }
5C4 }



E8D int main(){
52E   ios::sync_with_stdio(0);
C97   cin.tie(NULL);
14E   int n,m;
0E4   vector<vi>graph(ms);
E32   vector<int>topo;
AA3   cin >> n >> m;
7EC   vector <int> indegree (n+1,0);
DD5   for(int i = 0; i < m ; i++){
BA2     int a,b;
0A8     cin >> a >>b;
163     indegree[b]++;
8FA     graph[a].push_back(b);
397   }

8AE   bfs(graph,indegree,n,topo);

677   if(topo.size() != n){
8D7     cout <<"Sandro fails.";
A8F   }else{
603     for (int i = 0; i < n; i++){
3D6       cout << topo[i] << " ";
DD0     }
E5B   }

BB3   return 0;
B7A }
```

1.5 DijkstraCPHandbook

```
6F1 for (int i = 1; i <= n; i++) distance[i] = INF;
BB0 distance[x] = 0;
FFB q.push({0,x});
14D while (!q.empty()) {
A44   int a = q.top().second; q.pop();
48E   if (processed[a]) continue;
DA9   processed[a] = true;
A32   for (auto u : adj[a]) {
D5D     int b = u.first, w = u.second;
803     if (distance[a]+w < distance[b]) {
C49       distance[b] = distance[a]+w;
CF7       q.push({-distance[b],b}); // peso negativo pq e uma
    maxheap, se for fazer com min_heap colocar positivo
8A5     }
B4E   }
724 }
```

1.6 bfs

```
450 vis[x] = 1;
336 dist[x] = 0;
E7C q.push(x);
14D while(!q.empty()){
D15   int s = q.front(); q.pop();

942   for (auto u : adj[s]){
497     if(vis[u]) continue;
B9C     vis[u] = true;
408     dist[u] = dist[s] + 1;
F73     q.push(u);
173   }
84D }
```

1.7 bicolority

```
F79 #define pll pair<long long, long long>
0C1 #define vi vector<int>
358 #define vll vector<long long>
531 const int ms = 10e5 + 1;
0E4 vector <vi>graph(ms);
0D3 vector <bool> vis(ms,0);
712 bool auxi = 0; // 0 - 1    1 - 2
578 bool pssibe = 1;
4DB void dfs(int raiz, vector<int>&team,bool aux ){
2B9   vis[raiz] = 1;
A0D   if (!aux){
1A9     team[raiz] = 1;
417   }else{
1A4     team[raiz] = 2;
6CF   }
5E1   for (int v : graph[raiz]){
D90     if(!vis[v]){
818       dfs(v,team,!aux);
028     }else if (team[v] == team[raiz]){
3E9       pssibe = 0;
801     }
D04   }

F42 }

E8D int main(){
52E   ios::sync_with_stdio(0);
C97   cin.tie(NULL);
B5C   int m,n;
712   bool auxi = 0;
2DE   cin >> m >> n;
603   for (int i = 0; i < n ; i++){
BA2     int a,b;
0A8     cin >> a >>b;
8FA     graph[a].push_back(b);
4C6     graph[b].push_back(a);
9A3   }
71E   vector<int>team(m+1,0);
F61   for(int i = 1 ; i< m;i++){
A2B     if (vis[i] == 0){
7DC       dfs(i,team,auxi);
F7B     }
813   }

17A   if(!pssibe ){
1A3     cout << "IMPOSSIBLE";
F09   }else{
8EA     for (int i = 1; i <= m; i++)
F95     {
268       cout << team[i] << " ";
E58     }

B46   }
BB3   return 0;
44E }
```

1.8 floydwarshal

```
78A for( int i = 1; i <= n; i++){
160   for(int j = 1; j <= n; j++){
EA0     if (i== j) distance[i][j] = 0;
A96     else if(adj[i][j]) distance [i][j] = adj[i][j];
FB5     else distance[i][j]=INF;
F5C   }
16F }

5D3 for(int k =1; k<=n; k++){
78A   for( int i = 1; i <= n; i++){
160     for(int j = 1; j <= n; j++){
353       distance[i][j] = min (distance[i][j], distance[i][k
    ]+distance[k][j]);
408     }
222   }
AF2 }
```

2 numbertheory

2.1 maxprime

```
1F7 const int MAXC = 1e7 + 10;
BBB int maxp[MAXC];
73E #pragma GCC optimize("O3")

A63 void crivo(){
F29     for(int i=2; i<MAXC; i++)
```

```
F95     {
6C6         if(maxp[i]) continue;
1C4         maxp[i] = i;

30B         for(int j=i*2; j<MAXC; j += i)
9AA             maxp[j] = i;
E77     }
E2D }
```

2.2 MatrixExpo

```
D41 // Fast Exp
031 const ll mod = 1e9+7;


8D8 ll fexpll(ll a, ll n){
D54   ll ans = 1;
02A   while(n){
A19     if(n & 1) ans = (ans * a) % mod;
4E2     a = (a * a) % mod;
9D3     n >>= 1;
CAB   }
BA7   return ans;
D19 }
D41 // matriz quadrada
BE9 class Matrix{
673   public:
21E   vector<vector<ll>> mat;
2E6   int m;
1D7   Matrix(int m): m(m){
593     mat.resize(m);
2BC     for(int i = 0; i < m; i++) mat[i].resize(m,0);
809   }
215   Matrix operator * (const Matrix& rhs){
8EB     Matrix ans = Matrix(m);
94F     for(int i = 0; i < m; i++)
A75       for(int j = 0; j < m; j++)
800         for(int k = 0; k < m; k++)
1F7           ans.mat[i][j] = (ans.mat[i][j] + (mat[i][k] * rhs.mat[k][j]) % mod) % mod;
BA7     return ans;
2E6   }
A70 };

E2E Matrix fexp(Matrix a, ll n){
71E   int m = a.m;
8EB   Matrix ans = Matrix(m);
642   for(int i = 0; i < m; i++) ans.mat[i][i] = 1;
02A   while(n){
A50     if(n & 1) ans = ans * a;
476     a = a * a;
9D3     n >>= 1;
CDF   }
BA7   return ans;
966 }
```

2.3 Sieve

```
F79 #define pll pair<long long, long long>
0C1 #define vi vector<int>
358 #define vll vector<long long>

A8C const int LIM = 1e6 + 5;
BDE bool isPrime[LIM];

003 vi sieve(){ // crivo com os numeros primos
C08   memset(isPrime,1, sizeof(isPrime));
1F3   isPrime[0] = isPrime[1] = false;

AF0   vi primes;
C35   for(int i =2;i <LIM; i++){
5ED     if(isPrime[i]){
E74       primes.push_back(i);
889       for(int j = i * 2; j <LIM; j +=i){
90F         isPrime[j] = false;
8B7       }
00A     }
1C3   }

A20   return primes;
90C }


E8D int main(){
52E   ios::sync_with_stdio(0);
C97   cin.tie(NULL);

BB3   return 0;
4EC }
```

2.4 BinomioNewton

```
197 long long MOD = 1e9 + 7;

97E vll fact(2e5 +1,0);
E78 vll ifact(2e5 +1,0);

8D8 ll fexpll(ll a, ll n){
D54   ll ans = 1;
02A   while(n){
94A     if(n & 1) ans = (ans * a) % MOD;
49B     a = (a*a) % MOD;
9D3     n >>= 1;
ABB   }
BA7   return ans;
5DC }

1E0 ll mul(ll a, ll b){
568   return (a*b)%MOD;
142 }

FC9 ll bc(int n, int k){ // N choose K
ED2   if(n < k) return 0;
024   return fact[n] *ifact[k] % MOD * ifact[n-k] %MOD;
B9C }

342 div = fexpll(div,MOD-2);  <- inverso modular div

E8D int main(){
886   ios::sync_with_stdio(false);
B95   cin.tie(0);
0C0   fact[0] =1;
2E1   fact[1] = 1;
908   ifact[0] = fexpll(1,MOD-2);
306   for(int i =1; i <= 200000;i++){
DA2     fact[i] = (fact[i-1] * i) %MOD;
DBB     ifact[i] = fexpll(fact[i],MOD-2);
2C0   }
BB3   return 0;
D19 }
```

3 teoremas

3.1 Geometria

- **Fórmula de Euler:** Em um grafo planar ou poliedro convexo, temos: $V - E + F = 2$ onde V é o número de vértices, E o número de arestas e F o número de faces.
- **Teorema de Pick:** Para polígonos com vértices em coordenadas inteiras:

$$\text{Área} = i + \frac{b}{2} - 1$$

  onde i é o número de pontos interiores e b o número de pontos sobre o perímetro.
- **Teorema das Duas Orelhas (Two Ears Theorem):** Todo polígono simples com mais de três vértices possui pelo menos duas "orelhas"— vértices que podem ser removidos sem gerar interseções. A remoção repetida das orelhas resulta em uma triangulação do polígono.
- **Incentro de um Triângulo:** É o ponto de interseção das bissetrizes internas e centro da circunferência inscrita. Se a, b e c são os comprimentos dos lados opostos aos vértices $A(X_a, Y_a)$, $B(X_b, Y_b)$ e $C(X_c, Y_c)$, então o incentro (X, Y) é dado por:

$$X = \frac{aX_a + bX_b + cX_c}{a + b + c}, \quad Y = \frac{aY_a + bY_b + cY_c}{a + b + c}$$

- **Triangulação de Delaunay:** Uma triangulação de um conjunto de pontos no plano tal que nenhum ponto está dentro do círculo circunscrito de qualquer triângulo. Essa triangulação:
  - Maximiza o menor ângulo entre todos os triângulos.
  - Contém a árvore geradora mínima (MST) euclidiana como subconjunto.
- **Fórmula de Brahmagupta:** Para calcular a área de um quadrilátero cíclico (todos os vértices sobre uma circunferência), com lados a, b, c e d:

$$s = \frac{a + b + c + d}{2}, \quad \text{Área} = \sqrt{(s - a)(s - b)(s - c)(s - d)}$$

  Se $d = 0$ (ou seja, um triângulo), ela se reduz à fórmula de Heron:

$$\text{Área} = \sqrt{(s - a)(s - b)(s - c)s}$$

3.2 DP

- **Divide and Conquer Optimization:** Utilizada em problemas do tipo:

$$dp[i][j] = \min_{k<j}\{dp[i-1][k] + C[k][j]\}$$

onde o objetivo é dividir o subsegmento até j em i segmentos com algum custo. A otimização é válida se:

$$A[i][j] \leq A[i][j+1]$$

onde $A[i][j]$ é o valor de k que minimiza a transição.

- **Knuth Optimization:** Aplicável quando:

$$dp[i][j] = \min_{i<k<j}\{dp[i][k] + dp[k][j]\} + C[i][j]$$

e a condição de monotonicidade é satisfeita:

$$A[i][j-1] \leq A[i][j] \leq A[i+1][j]$$

com $A[i][j]$ sendo o índice k que minimiza a transição.

- **Slope Trick:** Técnica usada para lidar com funções lineares por partes e convexas. A função é representada por pontos onde a derivada muda, que podem ser manipulados com `multiset` ou `heap`. Útil para manter o mínimo de funções acumuladas em forma de envelopes convexos.
- **Outras Técnicas e Truques Importantes:**
  - **FFT (Fast Fourier Transform):** Convolução eficiente de vetores.
  - **CHT (Convex Hull Trick):** Otimização para DP com funções lineares e monotonicidade.
  - **Aliens Trick:** Técnica para binarizar o custo em problemas de otimização paramétrica (geralmente em problemas com limite no número de grupos/segmentos).
  - **Bitset:** Utilizado para otimizações de espaço e tempo em DP de subconjuntos ou somas parciais, especialmente em problemas de mochila.

3.3 Grafos

- **Fórmula de Euler (para grafos planares):**

$$V - E + F = 2$$

onde V é o número de vértices, E o número de arestas e F o número de faces.

- **Handshaking Lemma:** O número de vértices com grau ímpar em um grafo é par.
- **Teorema de Kirchhoff (contagem de árvores geradoras):** Monte a matriz M tal que:

$$M_{i,i} = \deg(i), \quad M_{i,j} = \begin{cases} -1 & \text{se existe aresta } i-j \\ 0 & \text{caso contrário} \end{cases}$$

O número de árvores geradoras (spanning trees) é o determinante de qualquer co-fator de M (remova uma linha e uma coluna).

- **Condições para Caminho Hamiltoniano:**
  - **Teorema de Dirac:** Se todos os vértices têm grau $\geq n/2$, o grafo contém um caminho Hamiltoniano.
  - **Teorema de Ore:** Se para todo par de vértices não adjacentes u e v, temos $\deg(u) + \deg(v) \geq n$, então o grafo possui caminho Hamiltoniano.
- **Algoritmo de Borůvka:** Enquanto o grafo não estiver conexo, para cada componente conexa escolha a aresta de menor custo que sai dela. Essa técnica constrói a árvore geradora mínima (MST).
- **Árvores:**
  - Existem C_n árvores binárias com n vértices (C_n é o n-ésimo número de Catalan).
  - Existem C_{n-1} árvores enraizadas com n vértices.
  - **Fórmula de Cayley:** Existem n^{n-2} árvores com vértices rotulados de 1 a n.
  - **Código de Prüfer:** Remova iterativamente a folha com menor rótulo e adicione o rótulo do vizinho ao código até restarem dois vértices.
- **Fluxo em Redes:**
  - **Corte Mínimo:** Após execução do algoritmo de fluxo máximo, um vértice u está do lado da fonte se $\text{level}[u] \neq -1$.
  - **Máximo de Caminhos Disjuntos:**
    - **Arestas disjuntas:** Use fluxo máximo com capacidades iguais a 1 em todas as arestas.
    - **Vértices disjuntos:** Divida cada vértice v em v_{in} e v_{out}, conectados por aresta de capacidade 1. As arestas que entram vão para v_{in} e as que saem saem de v_{out}.
  - **Teorema de König:** Em um grafo bipartido:

$$\text{Cobertura mínima de vértices} = \text{Matching máximo}$$

O complemento da cobertura mínima de vértices é o conjunto independente máximo.

  - **Coberturas:**
    - **Vertex Cover mínimo:** Os vértices da partição X que **não** estão do lado da fonte no corte mínimo, e os vértices da partição Y que **estão** do lado da fonte.
    - **Independent Set máximo:** Complementar da cobertura mínima de vértices.
    - **Edge Cover mínimo:** É $N-$matching, pegando as arestas do matching e mais quaisquer arestas restantes para cobrir os vértices descobertos.
  - **Path Cover:**
    - **Node-disjoint path cover mínimo:** Duplicar vértices em tipo A e tipo B e criar grafo bipartido com arestas de $A \to B$. O path cover é $N-$ matching.
    - **General path cover mínimo:** Criar arestas de $A \to B$ sempre que houver caminho de A para B no grafo. O resultado também é $N-$ matching.
  - **Teorema de Dilworth:** O path cover mínimo em um grafo dirigido acíclico é igual à **antichain máxima** (conjunto de vértices sem caminhos entre eles).
  - **Teorema do Casamento de Hall:** Um grafo bipartido possui um matching completo do lado X se:

$$\forall W \subseteq X, \quad |W| \leq |\text{vizinhos}(W)|$$

  - **Fluxo Viável com Capacidades Inferiores e Superiores:** Para rede sem fonte e sumidouro:
    - Substituir a capacidade de cada aresta por $c_{\text{upper}} - c_{\text{lower}}$
    - Criar nova fonte S e sumidouro T
    - Para cada vértice v, compute:

$$M[v] = \sum_{\text{arestas entrando}} c_{\text{lower}} - \sum_{\text{arestas saindo}} c_{\text{lower}}$$

    - Se $M[v] > 0$, adicione aresta (S, v) com capacidade $M[v]$; se $M[v] < 0$, adicione (v, T) com capacidade $-M[v]$.
    - Se todas as arestas de S estão saturadas no fluxo máximo, então um fluxo viável existe. O fluxo viável final é o fluxo computado mais os valores de c_{lower}.

3.4 Propriedades Matemáticas

- **Conjectura de Goldbach:** Todo número par $n > 2$ pode ser representado como $n = a + b$, onde a e b são primos.
- **Primos Gêmeos:** Existem infinitos pares de primos p, $p + 2$.
- **Conjectura de Legendre:** Sempre existe um primo entre n^2 e $(n+1)^2$.
- **Lagrange:** Todo número inteiro pode ser representado como soma de 4 quadrados.
- **Zeckendorf:** Todo número pode ser representado como soma de números de Fibonacci diferentes e não consecutivos.
- **Tripla de Pitágoras (Euclides):** Toda tripla pitagórica primitiva pode ser gerada por $(n^2 - m^2, 2nm, n^2 + m^2)$ onde n e m são coprimos e um deles é par.
- **Wilson:** n é primo se e somente se $(n-1)! \mod n = n - 1$.
- **Problema do McNugget:** Para dois coprimos x e y, o número de inteiros que não podem ser expressos como $ax + by$ é $(x-1)(y-1)/2$. O maior inteiro não representável é $xy - x - y$.
- **Fermat:** Se p é primo, então $a^{p-1} \equiv 1 \mod p$. Se x e m são coprimos e m primo, então $x^k \equiv x^{k \bmod (m-1)} \mod m$. *Euler:* $x^{\varphi(m)} \equiv 1 \mod m$. $\varphi(m)$ é o totiente de Euler.
- **Teorema Chinês do Resto:** Dado um sistema de congruências:

$$x \equiv a_1 \mod m_1, \quad \ldots, \quad x \equiv a_n \mod m_n$$

  com m_i coprimos dois a dois. E seja $M_i = \frac{m_1 m_2 \cdots m_n}{m_i}$ e $N_i = M_i^{-1} \mod m_i$. Então a solução é dada por:

$$x = \sum_{i=1}^{n} a_i M_i N_i$$

  Outras soluções são obtidas somando $m_1 m_2 \cdots m_n$.
- **Números de Catalan:** Exemplo: expressões de parênteses bem formadas. $C_0 = 1$, e:

$$C_n = \sum_{i=0}^{n-1} C_i C_{n-1-i} = \frac{1}{n+1}\binom{2n}{n}$$

- **Bertrand (Ballot):** Com $p > q$ votos, a probabilidade de sempre haver mais votos do tipo A do que B até o fim é: $\frac{p-q}{p+q}$ Permitindo empates: $\frac{p+1-q}{p+1}$. Multiplicando pela combinação total $\binom{p+q}{q}$, obtém-se o número de possibilidades.
- **Linearidade da Esperança:** $E[aX+bY] = aE[X]+bE[Y]$
- **Variância:** $\mathrm{Var}(X) = E[(X-\mu)^2] = E[X^2] - E[X]^2$
- **Progressão Geométrica:** $S_n = a_1 \cdot \frac{q^n - 1}{q - 1}$
- **Soma dos Cubos:** $\sum_{k=1}^{n} k^3 = \left(\sum_{k=1}^{n} k\right)^2$
- **Lindström-Gessel-Viennot:** A quantidade de caminhos disjuntos em um grid pode ser computada como o determinante da matriz do número de caminhos.
- **Lema de Burnside:** Número de colares diferentes (sem contar rotações), com m cores e comprimento n:

$$\frac{1}{n}\left(m^n + \sum_{i=1}^{n-1} m^{\gcd(i,n)}\right)$$

- **Inversão de Möbius:**

$$\sum_{d|n} \mu(d) = \begin{cases} 1, & n = 1 \\ 0, & \text{caso contrário} \end{cases}$$

- **Propriedades de Coeficientes Binomiais:**

$$\binom{N}{K} = \binom{N}{N-K} = \frac{N}{K}\binom{N-1}{K-1}$$

$$\sum_{k=0}^{m} (-1)^k \binom{n}{k} = (-1)^m \binom{n-1}{m}$$

$$\sum_{m=0}^{n} \binom{m}{k} = \binom{n+1}{k+1}$$

$$\sum_{k=0}^{m} \binom{n+k}{k} = \binom{n+m+1}{m}$$

$$\sum_{k=0}^{n} \binom{n}{k}^2 = \binom{2n}{n}$$

$$\sum_{k=0}^{n} \binom{n}{k} = 2^n$$

$$\sum_{k=0}^{n} k\binom{n}{k} = n \cdot 2^{n-1}$$

$$\sum_{k=0}^{n} \binom{n-k}{k} = F_{n+1}$$

- **Identidades Clássicas:**
  - **Hockey-stick:** $\sum_{i=r}^{n} \binom{i}{r} = \binom{n+1}{r+1}$
  - **Vandermonde:** $\binom{m+n}{r} = \sum_{k=0}^{r} \binom{m}{k}\binom{n}{r-k}$
- **Distribuições de Probabilidade:**
  - **Uniforme:** $X \in \{a, a+1, \ldots, b\}$, $E[X] = \frac{a+b}{2}$
  - **Binomial:** n tentativas com probabilidade p de sucesso:

$$P(X = x) = \binom{n}{x} p^x (1-p)^{n-x}, \quad E[X] = np$$

  - **Geométrica:** Número de tentativas até o primeiro sucesso:

$$P(X = x) = (1-p)^{x-1} p, \quad E[X] = \frac{1}{p}$$

4 seg

4.1 sum

```
2B7 #include<bits/stdc++.h>

F79 #define pll pair<long long, long long>
0C1 #define vi vector<int>
358 #define vll vector<long long>

AD1 typedef long long ll;

359 const int N = 2e5+7;
C8E int original[N];
0BB ll seg[4*N];
1A8 int n;

317 void build (int l =1,  int r = n, int idx=1){
893   if (l == r){
A0E     seg[idx] = original[l];
505     return;
98F   }
EE4   int m = (l + r) / 2;
B77   int left = idx * 2 ;
0E8   int right = idx * 2 + 1 ;
84F   build (l, m, left);
C3B   build (m +1,r, right);

5B2   seg[idx] = seg[left] + seg[right];
134 }

62A long long query(int ql,int qr,int l = 1, int r = n, int
     idx=1){
BDD   if (ql > r || qr < l) return 0;

E07   if (ql <= l && qr >= r){ // o l,r tem que estar
     obrigatoriamente dentro do range da query para poder
     contribuir, se tiver duvida irei desehar o que contribui
A9D     return seg[idx];
A78   }

EE4   int m = (l + r) / 2;
B77   int left = idx * 2 ;
0E8   int right = idx * 2 + 1 ;

496   return query(ql,qr,l,m,left) + query(ql,qr,m+1,r,right);

753 }

406 void update (int pos, ll val, int l=1, int r=n, int idx=1)
     {
```

```
893   if (l == r){
873     seg[idx] = val;
505     return;
741   }

EE4   int m = (l + r) / 2;
B77   int left = idx * 2 ;
0E8   int right = idx * 2 + 1;

E09   if (pos <= m){
3A1     update(pos,val,l,m,left);
896   }else{
8AD     update(pos,val,m+1,r,right);
7B0   }

5B2   seg[idx] = seg[left] + seg[right];
B5E }

E8D int main(){
52E   ios::sync_with_stdio(0);
C97   cin.tie(NULL);

834   int q; cin >> n >> q;

78A   for(int i = 1; i <= n; i++){
EA4     cin >> original[i];
85F   }

6F2   build();

646   for(int i = 1; i <= q; i++){
4C1     long long x,a,b;
6AA     cin >> x >> a >>b;
C91     if (x == 1){
E72       update(a,b);
FD6     }else{
E94       cout << query(a,b) << '\n';
CEB     }
6CC   }
BB3   return 0;
491 }
```

4.2 RangeUpdateQueries

```
2B7 #include<bits/stdc++.h>

F79 #define pll pair<long long, long long>
0C1 #define vi vector<int>
358 #define vll vector<long long>

AD1 typedef long long ll;

359 const int N = 2e5+7;
81E const int LIM = 1e4 + 10;

5B7 vector<ll> seg(4*N,0);
532 vector<ll> lazy(4*N,0); // o valor inicial da lazy varia
      tambem de acordo com o problema,
6F7 vll ar;
21C vi maxp(LIM,-1);
A31 vi sump(LIM,0);
4C3 vll original(N,0);
4DE ll n;

A35 void build(int l=1, int r = n, int no =1){
893   if (l == r){
446     seg[no] = original[l];
505     return;
43A   }
3E3   ll m = (l + r) / 2;
CBD   ll left = no * 2 ;
E81   ll right = no * 2 + 1 ;
84F   build (l, m, left);
C3B   build (m +1,r, right);

80C   seg[no] = seg[left] + seg[right];
041 }

072 void update_lazy(int no, int l, int r, ll v){
8D9   lazy[no] += v;  // essa atualizacao varia de acordo com
      o problema
5E1   seg[no] += ((r-l +1) * v);
C87 }

3B4 void propagate(int no, int l, int r) {
54D   if (lazy[no]  == 0) return;

579   if(l!=r){
EE4     int m = (l + r) / 2;
65B     update_lazy(2 * no, l, m, lazy[no]);
8D5     update_lazy(2 * no + 1, m + 1, r, lazy[no]);
AFE   }

099   lazy[no] = 0;
626 }

B5B void update( int ul, int ur, ll v, int no =1, int l=1, int
      r=n){

90C   if (r < ul || l > ur) return;
619   if (ul <= l && r <= ur) {
E30     update_lazy(no, l, r, v);
505     return;
DCD   }

483   propagate(no, l, r);

EE4   int m = (l + r) / 2;
C91   update( ul, ur, v,2 * no, l, m);
3C9   update( ul, ur, v, 2 * no + 1, m + 1, r);

027   seg[no] = seg[2 * no] + seg[2 * no + 1];
D23 }


A6C ll query(int L, int R,int l = 1, int r = n, int no = 1){

483   propagate(no,l,r);
1BA   if(R < l || L > r) return 0;
761   if(L <= l && r <=R){
B4F     return seg[no];
CA6   }
EE4   int m = (l+r)/2;
1F2   int left = 2*no;
176   int right = 2*no+1;
15D   ll saida  =query(L,R,l,m,left) + query(L,R,m+1,r,right);
B89   return saida;
69B }


E8D int main(){
52E   ios::sync_with_stdio(0);
C97   cin.tie(NULL);
3AA   cout.tie(NULL);
45F   int q;
55A   cin >>n>>q;
78A   for(int i = 1; i<= n;i++){
EA4     cin>>original[i];
85F   }
6F2   build();

ABF   for(int i = 0; i < q;i++){
E19     int ti; cin>>ti;
42D     if(ti == 1){
42C       ll a,b,x; cin >>a>>b>>x;
BD3       update(a,b,x);
095     }else{
3FE       int k; cin >>k;
A51       cout <<query(k,k)<<'\n';
8FA     }
C94   }



BB3   return 0;
65C }
```

5 Strings

5.1 Manachor

```
F79 #define pll pair<long long, long long>
0C1 #define vi vector<int>
358 #define vll vector<long long>

AD1 typedef long long ll;

E8D int main(){
52E   ios::sync_with_stdio(0);
C97   cin.tie(NULL);
905   string a;
964   cin >> a;
94D   int n = a.size();
BA7   vi d1(n);
358   vi d2(n);
3B8   for(int i = 0, l = 0, r = -1; i < n; i++){
DF9     int k;
6D2     if (i > r){
B24       k = 1;
260     }else{
CFC       k = min (d1[l + r -i], r - i + 1);
310     }
F96     while(0 <= i - k && i + k < n && a[i-k] == a[i + k]){
AC1       k++;
936     }
61E     d1[i] = k--;
17C     if(i + k > r){
BB6       l = i - k;
272       r = i + k;
007     }
E8E   }

3B8   for(int i = 0, l = 0, r = -1; i < n; i++){
DF9     int k;
6D2     if (i > r){
5A4       k = 0;
FAD     }else{
0E2       k = min (d2[l + r -i + 1], r - i + 1);
D6A     }
146     while(0 <= i - k -1 && i + k < n && a[i-k - 1] == a[i
      + k]){
AC1       k++;
```

```
BE7     }
E1C     d2[i] = k--;
17C     if(i + k > r){
56F       l = i - k -1;
272       r = i + k;
6C1     }
A15   }
C85   int max_par = 0;
805   int max_impar = 0;
478   int max_indice_par;
A41   int max_indice_impar;
724   for(int i = 0; i < d1.size();i++){
005     if(d1[i] >max_impar ){
788       max_indice_impar = i;
553     }
B2F     if(d2[i] >max_par ){
9BB       max_indice_par = i;
AD8     }
F8F     max_impar = max (max_impar,d1[i]);
D2A     max_par = max (max_par,d2[i]);
605   }
0BD   string ans;
370   if(max_par >= max_impar){
4F7     ans = a.substr(max_indice_par - max_par, max_par * 2);
1A6   }else{
4BB     ans = a.substr(max_indice_impar - (max_impar-1), max_impar * 2 -1 );
27E   }

F21   cout << ans;

BB3   return 0;
31E }
```

5.2 AhoCorasick

```
7CE using ll = long long int;
F79 #define pll pair<long long, long long>
0C1 #define vi vector<int>
358 #define vll vector<long long>

3C9 struct node { // pode adicionar um contador de prefixos
15E   node* p = nullptr;
05C   node* nxt[26] = {};
446   node* sl = nullptr; // suffix link
63A   node* ol = nullptr; // output link
0AF   int c;
919   int idx = -1; // -1 indica que nao eh fim de palavra.
FBA   int prf_cnt =0;
6D4 };


4BC typedef node* trie;
D41 //trie root = new node();

903 void add(trie root ,const string& s, int pattern_idx) { // O(S)
346   trie t = root;
BF4   t->prf_cnt++;
1CF   for (int i = 0; i < s.size(); i++) {
827     char c = s[i];
9AE     int v = c - 'a';
AC9     if (!t->nxt[v]) {
F90       trie son = new node();
EB1       son->c = v;
DAE       son->p = t;
B29       t->nxt[v] = son;
5A9     }
C80     t = t->nxt[v];
BF4     t->prf_cnt++;
C40   }

38C   t->idx = pattern_idx;
F6F }
B47 void buildLinks(trie root) { // O(L) L-> numero de nos

EF5   root->sl = root;

83D   queue<trie> q;
42F   for(int i = 0; i < 26; i++) {
280     if(root->nxt[i]) {
A94       q.push(root->nxt[i]);
FC8       root->nxt[i]->sl = root; // Filhos da raiz tem sl para a raiz
9D3     }
D8B   }

14D   while (!q.empty()) {
81D     trie t = q.front();
833     q.pop();

1E1     for (int c = 0; c < 26; c++) {
4B4       if (t->nxt[c]) {
371         trie son = t->nxt[c];
DD3         trie w = t->sl;
516         while (w != root && !w->nxt[c]) {
AA5           w = w->sl;
6B6         }
312         if (w->nxt[c]) {
BE9           son->sl = w->nxt[c];
6C5         } else {
F0A           son->sl = root;
BB5         }

D41         //LOGICA DO OUTPUT LINK
D41         // Verifica se o no do suffix link eh o fim de uma palavra.
D41         // Se for (idx != -1), o output link aponta para ele.
D41         // Senao, herda o output link dele.
C84         if (son->sl->idx != -1) {
AF1           son->ol = son->sl;
D2B         } else {
9C1           son->ol = son->sl->ol;
E1A         }

F96         q.push(son);
4F7       }
9E8     }
B93   }
E75 }

700 void search(trie root,const string& text, const vector<string>& patterns) { //O(M+Z) Custo para buscar em um texto de tamanho M com Z ocorrencias.

8B8   trie current = root;

D41   // 1. Percorre o texto um caractere por vez
1B3   for (int i = 0; i < text.length(); ++i) {
9CC     int v = text[i] - 'a';

D41     // 2. Logica de transicao: Se nao houver caminho direto,
D41     //    segue os suffix links (sl) ate encontrar um ou chegar na raiz.
F6A     while (current != root && !current->nxt[v]) {
B87       current = current->sl;
F50     }
723     if (current->nxt[v]) {
523       current = current->nxt[v];
8B3     }

D41     // 3. Verificacao de Padroes: Apos a transicao, verifica se ha
D41     //    algum padrao terminando nesta posicao.
B03     trie temp = current;
969     while (temp != nullptr) {
D41       // Se o no atual representa o fim de um padrao (idx != -1)...
924       if (temp->idx != -1) {
FED         int pattern_idx = temp->idx;
AB1         const string& found_pattern = patterns[pattern_idx];
D41         // Calcula a posicao inicial da palavra encontrada
218         int start_pos = i - found_pattern.length() + 1;
387         cout << "  -> Padrao '" << found_pattern << "' encontrado na posicao " << start_pos << endl;
381       }
D41       // ...pula para o proximo padrao na cadeia de sufixos usando o output link (ol).
80C       temp = temp->ol;
261     }
BBD   }
08C }


421 int cnt_prefix (trie root,const string& prefix){
3EC   for(auto c : prefix){
9AE     int v = c-'a';
93B     if(!root->nxt[v]){
BB3       return 0;
918     }
D73     root= root->nxt[v];
475   }
197   return root->prf_cnt;
08A }



E8D int main(){
B95   cin.tie(0);
52E   ios::sync_with_stdio(0);
C5F   int n,q; cin>>n>>q;
1F1   trie root = new node();


BB3   return 0;
9F1 }
```

5.3 Z

```
F79 #define pll pair<long long, long long>
0C1 #define vi vector<int>
358 #define vll vector<long long>

AD1 typedef long long ll;

07E vector<int> z(string &s){
163   int n= s.size();
BBF   vector<int>zf(n,0);
898   int r = 0;
B8D   int l = 0;
6F5   for(int i = 1; i < n; i++){
4F3     if(i <= r){
DF4       zf[i] = min(zf[i-l], r - i + 1);
```

```
A58     }

118     while(s[zf[i]] == s[i + zf[i]] && i + zf[i] < n){
547       zf[i]++;
342     }

EC5     if (i + zf[i] -1 > r){ // o intervalo q eu olhei passou de r, eu atualizo o r pois ele maraca ate quando olhei
A89       r = i + zf[i] -1;
537       l = i;
D41     }
424   }
A6D   return zf;
124 }

E8D int main(){
52E   ios::sync_with_stdio(0);
C97   cin.tie(NULL);
59B   string n,m;
AA3   cin >> n >> m;
053   string nova = m + '$';
0BA   nova += n;
80C   vi ans = z(nova);
3BD   int resp = 0;
C3E   for(int i = 0; i < ans.size();i++){
B0A     if (ans[i] == m.size()){
7AD       resp++;
19B     }
0EF   }
046   cout << resp;


BB3   return 0;
EEA }
```

5.4 hash

```
F79 #define pll pair<long long, long long>
0C1 #define vi vector<int>
358 #define vll vector<long long>

AD1 typedef long long ll;

E8E const ll MOD1 = 131'807'699;
0B3 const ll MOD2 =127'065'427;
0C8 const ll base = 127; //127
80E const int MAXN = 1e5 + 5; // tamanho da string
D41 /*
097 Some Big Prime Numbers:
E47 37'139'213
99A 127'065'427
56A 131'807'699
C4C */

651 ll expBase1[MAXN];
1B2 ll expBase2[MAXN];

FE8 void precalc(){
243   expBase1[0]=expBase2[0]=1;
B74   for(int i=1;i<MAXN;i++){
F7C     expBase1[i] = (expBase1[i-1]*base)%MOD1;
26F     expBase2[i] = (expBase2[i-1]*base)%MOD2;
FF9   }
BF5 }

332 struct PolyHash{
0DD   vector<pair<ll,ll>> hsh;

C90   PolyHash(string& s){
20A     hsh = vector<pair<ll,ll>> (s.size()+1,{0LL,0LL});
1CF     for(int i=0;i<s.size();i++){
1AB       hsh[i+1].first = ( (hsh[i].first *base) % MOD1 + s[i] ) % MOD1;
9DD       hsh[i+1].second =( (hsh[i].second*base) % MOD2 + s[i] ) % MOD2;
213     }
A7A   }

2F0   ll gethash(int a, int b){
4FC     ll h1 = (MOD1 + hsh[b+1].first  - ( hsh[a].first * expBase1[b-a+1] ) % MOD1) % MOD1;
695     ll h2 = (MOD2 + hsh[b+1].second - ( hsh[a].second* expBase2[b-a+1] ) % MOD2) % MOD2;
A46     return (h1<<32LL) | h2;
842   }
ADD };
```

5.5 RabinKarp

```
D41 //source:https://cp-algorithms.com/string/rabin-karp.html

295 vector<int> rabin_karp(string const& s, string const& t) {
E96     const int p = 31;
3B1     const int m = 1e9 + 9;
2B7     int S = s.size(), T = t.size();

A00     vector<long long> p_pow(max(S, T));
B60     p_pow[0] = 1;
EFD     for (int i = 1; i < (int)p_pow.size(); i++)
1CA         p_pow[i] = (p_pow[i-1] * p) % m;

976     vector<long long> h(T + 1, 0);
6E9     for (int i = 0; i < T; i++)
D4F         h[i+1] = (h[i] + (t[i] - 'a' + 1) * p_pow[i]) % m;
554     long long h_s = 0;
109     for (int i = 0; i < S; i++)
161         h_s = (h_s + (s[i] - 'a' + 1) * p_pow[i]) % m;

A15     vector<int> occurrences;
FCE     for (int i = 0; i + S - 1 < T; i++) {
F0F         long long cur_h = (h[i+S] + m - h[i]) % m;
D52         if (cur_h == h_s * p_pow[i] % m)
879             occurrences.push_back(i);
F05     }
831     return occurrences;
5E2 }
```

6 dp

6.1 coincombination

```
F79 #define pll pair<long long, long long>
0C1 #define vi vector<int>
358 #define vll vector<long long>

AD1 typedef long long ll;

F53 const int MAXN = 1000001;
D41 //int dp[MAXN][MAXN];
2C3 const int MODULO = 1e9 + 7;
C08 vll coins;
1A6 ll dp[MAXN];

E8D int main(){
52E   ios::sync_with_stdio(0);
C97   cin.tie(NULL);
CBB   ll n, x;
B8A   cin >> n >> x;
F0D   memset(dp,0, sizeof dp);
619   coins.resize(n);
603   for (int i = 0; i < n; i++){
283     cin >> coins[i];
84D   }
0EC   dp[0] = 1;

218 for (int i = 1; i <= x; i++){
6B9   for(auto c: coins){
D4C     if (i >= c){
C70       dp[i] = dp[i] + dp[i-c];
01B       dp[i] = dp[i] % MODULO;
B63     }
5DD   }
D6C }
1D5   cout << dp[x];
BB3   return 0;
681 }
```

6.2 coin

```
F79 #define pll pair<long long, long long>
0C1 #define vi vector<int>
358 #define vll vector<long long>

AD1 typedef long long ll;

D41 //const int MAXN = 3001;
D41 //int dp[MAXN][MAXN];

E8D int main(){
52E   ios::sync_with_stdio(0);
C97   cin.tie(NULL);
CBB   ll n, x;
B8A   cin >> n >> x;
C08   vll coins;

603   for (int i = 0; i < n; i++){
40A     ll a; cin >> a;
541     coins.push_back(a);
6A3   }

7E0   vll dp (x+1,1e9);
076   dp[0] = 0;
6B9   for (auto c : coins){
D55     for (int i = c; i <= x; i++){
314       dp[i] = min (dp[i],dp[i-c] +1 );
BEC     }
51A   }
15E   if (dp[x]==1e9 ){
866     cout << -1;
ECB   }else{
1D5     cout << dp[x];
FAD   }


BB3   return 0;
C9B }
```

6.3 LIS

```
433 int lis() {
471   vector<int> dp;
941   for (int i : a) {
01D     int pos = lower_bound(dp.begin(), dp.end(), i) - dp.begin();
CD1     if (pos == dp.size()) {
D8F       dp.push_back(i);
793     } else {
0BB       dp[pos] = i;
AB0     }
CB8   }
269   return dp.size();
B71 }
```

6.4 recupera

```
797 vll a;

1DA const int sumax = 100001;
97B const int nmax = 101;

DDF int dp [nmax][sumax];
267 int extra [nmax][sumax];


030 int solve(int n, int falta){

8E0   if(n == a.size()) return 0;

713   if(falta == 0) return 1;

CB7   if(dp[n][falta] != -1) return dp[n][falta];

997   int op1,op2;

E45   op1 = solve(n+1,falta- a[n]); // pegar
4E0   op2 = solve(n+1,falta); // nao pegar

4C6   if(op1 == 1){
99F     extra[n][falta] = 1;
E85   }else{
F5C     extra[n][falta] = 0;
9BD   }

F9F   dp[n][falta] = (op1||op2);

A17   return dp[n][falta];

7C7 }
```

6.5 knapsack

```
F79 #define pll pair<long long, long long>
0C1 #define vi vector<int>
358 #define vll vector<long long>

AD1 typedef long long ll;

D41 //const int MAXN = 3001;
D41 //int dp[MAXN][MAXN];

E8D int main(){
52E   ios::sync_with_stdio(0);
C97   cin.tie(NULL);
949   int n, w;
DE5   cin >> n >> w;

6B8   vector<pii> entrada;
603   for(int i = 0 ; i < n; i++){
BA2     int a,b;
0A8     cin >> a >> b;
EB8     entrada.push_back(make_pair(a,b));
6E6   }


926   vector<vll>dp(n + 1,vll(w+1,0));

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

B55     int peso = entrada[i].first;
6EC     int valor = entrada[i].second;
940     if(i > 0){
D0F     for (int x = 0; x < w; x++){
864         dp[i+1][x+1] = dp[i][x+1];
9E3     }
3CE     }

7B7     for(int j = 0; j < w ; j++){
7B6       if(j+1>= peso){
599         dp[i+1][j+1] = max(dp[i][j+1], dp[i][j-peso +1]+valor );
AD8       }


C78     }

89F   }

CB3   cout << dp[n][w];



BB3   return 0;
4DB }
```

6.6 LCS

```
F79 #define pll pair<long long, long long>
0C1 #define vi vector<int>
358 #define vll vector<long long>

AD1 typedef long long ll;

D41 //const int MAXN = 3001;
D41 //int dp[MAXN][MAXN];

E8D int main(){
52E   ios::sync_with_stdio(0);
C97   cin.tie(NULL);
AC1   string s,t;
FEE   cin >> s >>t;
F3F   vector<vi>dp(s.size() + 1,vi(t.size()+1,0));
CA5   int max_sub = 0;
BAC   int pos_s;
E42   for (int i = 1; i <=s.size();i++ ){
F69     dp[i][0] = 0;
631     for (int j = 1; j <= t.size();j++){
CC3       if(t[j-1] == s[i-1]){
0F6         dp[i][j] = 1 + dp[i-1][j-1] ;
DF7       }else{
398         dp[i][j] = max (dp[i-1][j], dp[i][j-1]);
A96       }
3E9     }
93F   }
34C   int aux = 0;
E29   int v = 0;
163   int n = s.size();
FA8   int m = t.size();
0BD   string ans;
E55   while(n > 0 && m > 0){
060     if (dp[n][m] == dp[n][m-1]){
76B       m--;
BF4     }else if(dp[n][m] == dp[n-1][m]){
15A       n--;
074     }else{
15A       n--;
4F1       ans = ans + s[n];
76B       m--;
7E6     }
F21   }

459   reverse(ans.begin(),ans.end());

F21   cout << ans;


BB3   return 0;
05A }
```

7 Geometry

7.1 PointInside

```
7CE using ll = long long int;
B73 #define pff pair<float, float>
F79 #define pll pair<long long, long long>
0C1 #define vi vector<int>
358 #define vll vector <long long>
E1F #define INF 0x3f3f3f3f

B2A struct pt {
0BE   ll x, y;
D30   pt() : x(0), y(0) {}
0FA   pt(ll x, ll y) : x(x), y(y) {}
C17   pt operator + (pt o){ return pt(x + o.x, y + o.y);}
55F   pt operator - (pt o){ return pt(x-o.x, y-o.y ); }
A14   pt operator *(ll k){ return pt(k*x, k*y ); }
9BF   ll len(){return hypot(x,y); }
8C8   ll cross(pt o) {return x* o.y - y* o.x ;}
B42   bool operator == (pt o) {return tie(x,y) == tie(o.x,o.y); }
F41   bool operator != (pt o) {return tie(x,y) != tie(o.x,o.y); }
357   bool operator < (pt o) {return tie(x,y) < tie(o.x,o.y); }

2A4 };
AC7 int sgn(long long val) { return val > 0 ? 1 : (val == 0 ? 0 : -1); }

043 bool lexComp(const pt &l, const pt &r) {
47C     return l.x < r.x || (l.x == r.x && l.y < r.y);
EF9 }

891 vector<pt> seq;
99B pt translation;
1A8 int n;
```

```
7A7 bool ptInTriangle(pt a, pt b, pt c, pt point) {
0E1   ll s1 = abs((b-a).cross(c-b));
708   ll area1 = abs((point - a).cross(point - b));
CC2   ll area2 = abs((point - b).cross(point - c));
243   ll area3 = abs((point - c).cross(point - a));
7F8   ll s2 = area1 + area2 + area3;
83F   return s1 == s2;
889 }
2B3 void prepare(vector<pt> &points) {
2BC     n = points.size();
BEC     int pos = 0;
6F5     for (int i = 1; i < n; i++) {
B9B         if (lexComp(points[i], points[pos]))
E4C             pos = i;
9F3     }
CF4     rotate(points.begin(), points.begin() + pos, points.end());
15A     n--;
317     seq.resize(n);
830     for (int i = 0; i < n; i++)
37D         seq[i] = points[i + 1] - points[0];
D86     translation = points[0];
837 }

E1D bool ptInConvexPolygon(pt ponto) {
216   ponto = ponto - translation;
692   if (seq[0].cross(ponto) != 0 &&sgn(seq[0].cross(ponto)) != sgn(seq[0].cross(seq[n - 1])))){
D1F     return false;
B3C   }
5C9   if (seq[n - 1].cross(ponto) != 0 &&sgn(seq[n - 1].cross(ponto)) != sgn(seq[n - 1].cross(seq[0])))){
D1F     return false;
5F1   }
28F   if (seq[0].cross(ponto) == 0){
9F5     return seq[0].len() >= ponto.len();
4F1   }
561   int l = 0, r = n - 1;
219   while (r - l > 1) {
AE0     int mid = (l + r) / 2;
351     int pos = mid;
563     if (seq[pos].cross(ponto) >= 0){
229       l =mid;
177     }else{
168       r = mid;
66F     }
5C7   }
5AB   int pos = l;
526   return ptInTriangle(seq[pos], seq[pos + 1], pt(0, 0), ponto);
BB8 }
```

7.2 ch

```
B2A struct pt {
662   double x, y;
D30   pt() : x(0), y(0) {}
E47   pt(double x, double y) : x(x), y(y) {}
C17   pt operator + (pt o){ return pt(x + o.x, y + o.y);}
55F   pt operator - (pt o){ return pt(x-o.x, y-o.y ); }
3F5   pt operator *(double k){ return pt(k*x, k*y ); }
F7E   double len(){return hypot(x,y); }
811   double cross(pt o) {return x* o.y - y* o.x ;}
B42   bool operator == (pt o) {return tie(x,y) == tie(o.x,o.y); }
F41   bool operator != (pt o) {return tie(x,y) != tie(o.x,o.y); }
357   bool operator < (pt o) {return tie(x,y) < tie(o.x,o.y);
  }

742 };

458 int orientation(pt a, pt b, pt c) {
2B5   pt AB = b-a;
0D1   pt BC = c-b;
FCC   double v = AB.cross(BC);
0E9   if (v < 0) return -1; // clockwise
896   if (v > 0) return +1; // counter-clockwise
BB3   return 0;
6F8 }

D90 bool cw(pt a, pt b, pt c, bool include_collinear) {
6EE   int o = orientation(a, b, c);
FFD   return o < 0 || (include_collinear && o == 0);
1D6 }
13D bool ccw(pt a, pt b, pt c, bool include_collinear) {
6EE   int o = orientation(a, b, c);
965   return o > 0 || (include_collinear && o == 0);
926 }

628 void convex_hull(vector<pt>& a, bool include_collinear = false) {
53D   if (a.size() == 1){
505     return;
C94   }
D41   //sort(a.begin(),a.end());  if!sorted
013   pt p1 = a[0], p2 = a.back();
4DD   vector<pt> up, down;
1CB   up.push_back(p1);
726   down.push_back(p1);
97F   for (int i = 1; i < (int)a.size(); i++) {
9A6     if (i == a.size() - 1 || cw(p1, a[i], p2, include_collinear)) {
A6E         while (up.size() >= 2 && !cw(up[up.size()-2], up[up.size()-1], a[i], include_collinear)){
B1C           up.pop_back();
DDA         }
F29       up.push_back(a[i]);
48C     }
8F4     if (i == a.size() - 1 || ccw(p1, a[i], p2, include_collinear)) {
80E       while (down.size() >= 2 && !ccw(down[down.size()-2], down[down.size()-1], a[i], include_collinear)){
ABC         down.pop_back();
56D       }
48A       down.push_back(a[i]);
2EC     }
F64   }
066   if(include_collinear && up.size() == a.size()) {
3F7     reverse(a.begin(), a.end());
505     return;
C18   }
228   a.clear();
775   for (int i = 0; i < (int)up.size(); i++){
DED     a.push_back(up[i]);
647   }
B5A   for (int i = down.size() - 2; i > 0; i--){
F3F   a.push_back(down[i]);
FF7   }
55E }
A17 double dist(pt a,pt b){
F30   double dist = (a.x - b.x)*(a.x - b.x) + (a.y - b.y)*(a.y - b.y);
672   return (sqrt(dist));
943 }
```

8 Extra

8.1 Hash Function

```
Call
```

```
g++ hash.cpp -o hash
./hash < code.cpp
```

```
to get the hash of the code.

The hash ignores comments and whitespaces.
The hash of a line whith } is the hash of all the code since
    the { that opens it. (is the hash of that context)

(Optional) To make letters upperCase: for(auto&c:s)if('a'<=c)
    c^=32;
```

```
DE3 string getHash(string s){
909   ofstream ip("temp.cpp"); ip << s; ip.close();
EE9   system("g++ -E -P -dD -fpreprocessed ./temp.cpp | tr -d '[:space:]' | md5sum > hsh.temp");
CEF   ifstream fo("hsh.temp"); fo >> s; fo.close();
A15   return s.substr(0, 3);
17A }

E8D int main(){
973   string l, t;
3DA   vector<string> st(10);
C61   while(getline(cin, l)){
54F     t = l;
242     for(auto c : l)
F11       if(c == '{') st.push_back(""); else
2F0       if(c == '}') t = st.back() + l, st.pop_back();
C33     cout << getHash(t) + " " + l + "\n";
1ED     st.back() += t + "\n";
D1B   }
B65 }
```