1. Hình học
   A(x1,y1), B(x1, y1)
   F(x,y) = a.x + b.y + c
   - A.B = |A| * |B| cos(A,B)
   - AxB = |A| * |B| sin(A,B) (2D)
   - Phương trình đường thẳng
   - (y1 - y2).x + (x2 - x1).y +(x1.y2 - x2.y1) = 0
   - Khoảng cách từ (xo, yo) đến đường thẳng a.x + b.y + c= 0
   - d = (a.xo + b.yo + c)/ sqrt(a.a + b.b)
   - A, B nằm khác phía với nhau qua đường thẳng F: F(x1,y1) * F(x2,y2) < 0
   - Vị trí tương đối 2 đường thẳng
     d1: a1.x + b1.y + c1= 0; d2: a2.x + b2.y + c2= 0
     D = a1.b2 - a2.b1
     Dx= b1.c2 - b2.c1
     Dy= a1.c2 - a2.c1
     - Nếu D != 0 thì 2 đường thẳng cắt nhau tại (xo, yo) = (Dx/D, Dy/D)
     - Nếu D == 0
       - Nếu Dx != 0 hoặc Dy != 0 : Song song
       - Nếu Dx = Dy = 0: Trùng nhau
   - Rẽ trái , phải
   - cho a,b,c và đang đi hướng từ a -> b hỏi để từ b -> c cần rẽ trái hya phải
     - K = (xb - xa)(yc - yb) - (yb - ya)(xc - xb)
       - K < 0: Rẽ trái
       - K > 0: Rẽ phải
       - K == 0: Đi thẳng
   - Tam giác
   - Diện tích S = sqrt(p.(p-a)(p-b)(p-c)) = (b.c.sinA)/2 = p.r = a.b.c/(4R)
2. Số học
   - Tổng cấp số nhân: Sn = (1- q^n)/(1-q)
   - Tổng cấp số cộng: Sn = n.(n-1).d/2
   - Tổ hợp: C(n,k) = (n!)/((n-k)! * k!)
   - Tổ hợp DP:
     - C(n,0) = C(n,n) = 1
     - C(n,k) = C(n-1,k-1) + C(n-1,k)
   - Chỉnh hợp: A(n,k) = n!/(n-k)!
   - Ước số, bội số:
     - N = a^i. b^j. ... c^k
       - Số ước của N: (i+1)(j+1)..(k+1)
       - Tổng các ước của N: (a^(i+1)-1)/(a-1) . (b^(j+1)-1)/(b-1) . ... (c^(k+1)-1)/(c-1)
   - Số mũa của số nguyên tố p trong n! là Sigma(1..k)([n/p^i]
   - Cho dãy a1, a2, a3 ... an, X là số thứ nhỏ thứ (n/2 + 1) thì F(X) = |a1 - X| + |a2 - X| + ...+ |an - X| đạt min

- GCD extended

```
void gcd_extended(int a, int b, int &x, int &y, int &d){
 int u1,u2,u3,v1,v2,v3,t1,t2,t3;
 u1 = 1; u2= 0; u3 = a;
 v1 = 0; v2= 1; v3 = b;
 while(v3 != 0){
     int q = u3/v3;
     t1 = u1 - q*v1;
     t2 = u2 - q*v2;
     t3 = u3 - q*v3;
     u1 = v1; u2= v2; u3= v3;
     v1 = t1; v2= t2; v3= t3;
 }
 x = u1; y = u2; d = u3;
}
```

3. Đồ thị

- Dijkstra

```
int dijktra(int s, int t){
   dist.resize(n+1); pre.resize(n+1);
   visited.resize(n+1);
   for(int i = 1; i<= n ;i++){
       dist[i]= oo;
       pre[i] = s;
       visited[i] = 0;
   }
   dist[s] = 0;
   heap.push(edge(s,0));
   while(!heap.empty()){
       edge e = heap.top(); heap.pop();
       int u = e.u;
       if(u == t) break;
       if(visited[u]) continue;
       for(int i= 0;i < dsk[u].size(); i++){
           int v,w;
           v = dsk[u][i].u;
           w = dsk[u][i].w;
           if(visited[v]) continue;
           if(dist[v] > dist[u] + w){
               dist[v] = dist[u] + w;
               pre[v] = u;
               heap.push(edge(v,dist[v]));
           }
       }
       visited[u] = 1;
   }
   return dist[t];
}
```

- Kruskal

```
struct edge{
   int u, v, w;
   void Print(){
       cout<<u<<" "<<v<<" "<<w<<endl;
   }
};
bool operator < (edge e1, edge e2){return e1.w > e2.w;}
priority_queue < edge > Heap; // chứa danh sach cạnh
vector < int > Par;
vector < int > Rank;
vector < edge > Tree;
int n,m;// n: số đỉnh, m số cạnh
void init(){
   Par.resize(n+1);
   Rank.resize(n+1);
   for(int i=1;i<=n;i++){
       Par[i] = i;
       Rank[i] = 0;
   }
}
int GetRoot(int r){
   while(Par[r] != r) r = Par[r];
   return r;
}
void Union(int r1, int r2){
   // r1, r2: root
   if(Rank[r1] > Rank[r2])
       Par[r2] = r1;
   else
       if(Rank[r1] < Rank[r2])
           Par[r1] = r2;
       else{
           Par[r2] = r1;
           Rank[r1]++;
       }
}
bool kruskal(){
   int r1, r2;
   edge e;
   while(!Heap.empty()){
       e = Heap.top(); Heap.pop();
       r1 = GetRoot(e.u);
       r2 = GetRoot(e.v);
       if(r1 != r2){
           Tree.push_back(e);
           if(Tree.size()== n-1) return true;
           Union(r1, r2);
       }
   }
   return false;
}
```

- Tazjan

```
int n, m;
vector<int> a[10005];
int cnt = 0, low[10005], num[10005], isPoint[10005];
int points = 0, bridges = 0;
void dfs(int u, int p) {
   int children = 0;
   num[u] = low[u] = cnt++;
   for(int v : a[u]) {
       if (num[v] == -1) {
           children++;
           dfs(v, u);
           // u "may" be articulation point
           if (low[v] >= num[u])
               isPoint[u] = (u == p) ? (children > 1) : 1;
           // u-v is bridges
           if (low[v] > num[u])
                bridges++;
           low[u] = min(low[u], low[v]);
       } else if (v != p)
           low[u] = min(low[u], num[v]);
   }
}
```

- Floyd

```
for (i=1; i<=n; i++)
       for (j=1; j<=n; j++)
               a[i][j] = oo;
for (i=1; i<=n; i++)
       a[i][i] = 0;
for (i=1; i<=m; i++){
       cin>>p>>q>>w;
       a[p][q] = a[q][p] = w;
}
for (k=1; k<=n; k++)
       for (i=1; i<=n; i++)
               for (j=1; j<=n; j++)
                       a[i][j] = min(a[i][j], a[i][k] + a[k][j]);
```

- Euler Tour

```
void dfs(int u, int &indx)
{
   vis[u] = 1;
   Euler[indx++] = u;
   for (auto it : adj[u]) {
       if (!vis[it]) {
           dfs(it, indx);
           Euler[indx++] = u;
       }
   }
}
```

- Kĩ thuật

- Mảng cộng dồn 2D
  - F[i][j] = F[i-1][j] + F[i][j-1] + F[i][j] - F[i-1][j-1]
  - Get(x1, y1, x2, y2) = F[x2][y2] - F[x2][y1-1] - F[x1 - 1][y2] + F[x1-1][y1-1]
- Rời rạc hóa:

```
for(int i=0; i<n;i++){
     v[i].first = a[i];
     v[i].second = i;
 }
 sort(v.begin(),v.end());
 int d = 1;
 for(int i=0; i<n;i++){
     b[v[i].second] = d;
     if(v[i+1].first > v[i].first ) d++;
 }
```

- Tìm kiếm tam phân

```
double max_f(double left, double right) {
   int N_ITER = 100;
   for (int i = 0; i < N_ITER; i++) {
       double x1 = left + (right - left) / 3.0;
       double x2 = right - (right - left) / 3.0;
       if (f(x1) > f(x2)) right = x2;
       else left = x1;
   }
   return f(left);
}
```

- Sàng nguyên tố

```
void sieve(int N) {
   bool isPrime[N+1];
   for(int i = 0; i <= N;++i)
       isPrime[i] = true;
   isPrime[0] = false;
   isPrime[1] = false;
   for(int i = 2; i * i <= N; ++i) {
       if(isPrime[i] == true) {i
           // Mark all the multiples of i as composite numbers
           for(int j = i * i; j <= N; j += i)
               isPrime[j] = false;
       }
   }
}
```

- ϕ(N) : Số lượng số nguyên tố cùng nhau với N
  - `ϕ(N)=n*∏(1−1/p)` với p là các ước nguyên tố cuả N
- Nghịch đảo modulo m của d là x khi $xd + my = 1$ (gcd_extended)
- Bao lồi

```
void convexHull(point[] X, boolean onEdge)
```

```
{
    int N = lengthof(X);
    int p = 0;
    boolean[] used = new boolean[N];
    //First find the leftmost point
    for (int i = 1; i < N; i++)
    {
        if (X[i] < X[p])
            p = i;
    }
    int start = p;
    do
    {
        int n = -1;
        int dist = onEdge ? INF : 0;
        for (int i = 0; i < N; i++){
            //X[i] is the X in the discussion
            //Don't go back to the same point you came from
            if (i == p)
                continue;
            //Don't go to a visited point
            if (used[i])
                continue;
            //If there is no N yet, set it to X
            if (n == -1)
                n = i;
            int cross = (X[i] - X[p]) x(X[n] - X[p]);
            //d is the distance from P to X
            int d = (X[i] - X[p]) · (X[i] - X[p]);
            if (cross < 0){
                //As described above, set N=X
                n = i;
                dist = d;
            }
            else if (cross == 0){
                //In this case, both N and X are in the
                //same direction. If onEdge is true, pick the
                //closest one, otherwise pick the farthest one.
                if (onEdge && d < dist)
                {
                    dist = d;
                    n = i;
                }
                else if (!onEdge && d > dist)
                {
                    dist = d;
                    n = i;
                }
            }
        }
        p = n;
        used[p] = true;
    } while (start != p);
}
```