Suffix Array

```
namespace SA{
#define XX w[ok]
#define YY w[!ok]
int SA[MAXN],cnt[256+1], w[3][MAXN];
int N,i,pot,p,k,ok, range;

void radix_pass(int *a,bool s){
   memset(cnt,0, range * 4);
   FOR(i,N) cnt[ a[i] ]++;
   FAB(i,1,range) cnt[i] += cnt[ i-1 ];
   FORR(i,N-1) SA[ --cnt[a[i]]]= s? YY[i]:i;
}
void Sufix_Array(char *cad){
   N = strlen(cad)+1, ok=0, range = 256;
   for(i=0;i<N;i++) XX[i] = cad[i];
   radix_pass(XX,0);
   for(int pot=p=1; p<N; pot *=2, range=p){
      for(p=0,i=N-pot; i<N; YY[p++]= i++) ;
      for(i=0;i<N;i++) if (SA[i] >= pot)
         YY[p++]= SA[i]-pot;
      for(i=0;i<N;i++) w[2][i]= XX[YY[i]];
      radix_pass(w[2],1);
      ok ^= p=1, XX[SA[0]] = 0;
      for(i=1;i<N; XX[SA[i++]]=k? p-1:p++)
         if(k=0, YY[SA[i]] == YY[SA[i-1]])
            if(YY[SA[i]+pot] == YY[SA[i-1]+pot])
               k = 1;
    }
}

int rank[MAXN], LCP[MAXN];
void findLCP(char *cad,int N){
   int i, j, k;
   for(i=1;i <= N;i++) rank[ SA[i] ] = i;
   for(k=i=0; i< N; LCP[rank[i++]]=k ,k-=k>0)
      for(j=SA[rank[i]-1];cad[i+k]==cad[j+k];k++);
}
};
```

Manacher

```
int rad[2*MAXN];
void Manacher(char *s,int n){
   int i=0,j=0,k;
   while(i < 2 * n - 1 ) {
      while(i >= j && i+j+1< 2*n &&
          s[(i-j)/ 2]==s[(i+j+1)/2])
         j++;
      rad[i] = j, k = 1;
      while(k <=rad[i] && rad[i-k]!=rad[i]-k){
         rad[i+k ]=min(rad[i-k],rad[i]-k);
         k++;
      }
      j = max(j-k,0),i +=k;
   }
}
```

Z-Algorithm

```
int Z[MAXN]; // Z[i]=SA[i]%SA[0]
void zAlgorithm(char *S,int n){
   int g=0,f=0; Z[0] = n;
   FAB(i,1,n)
      if(i<g && Z[i-f]!=(g-i))
         Z[i]=min(Z[i-f],g-i);
      else{
         g = max(g, f = i);
         while(g<n && S[g]==S[g-f]) g++;
         Z[i] = g - f;
      }
}
```

Aho Corasick Matrix

```
void Solve() {
   N=0; FOR(i,size+1) if(!terminal[i]) ID[i]=N++;
   for(int u,v = 0; v<=size;v++)
      if (!terminal[v]) FOR(c,alfa)
         if (!terminal[u = next[v][c]])
            MAT[ ID[v] ][ ID[u] ]++;
}
```

Suffix Automaton

```
int root=1,size[MAXN],next[MAXN][ALFA],fail[MAXN];
bool is_clone[MAXN];
int firstpos[MAXN], cnt[MAXN];

int SAsize,last;
void sa_init(){
   SAsize = last = 1;
   fail[1] = size[1] = 0;
   //memset(next,0,sizeof(next));
}
void sa_extend(char c){
   int now,cpy,w,car = CONV(c);
   next[last][car] = now = ++SAsize;
   size[now] = size[last] + 1;
   cnt[now] = 1, is_clone[now] = false;
   firstpos[now] = size[now] - 1;
   w = fail[last];
   while(w && !next[w][car])
      next[w][car]= now,w =fail[w];
   int q = next[w][car];
   if(w == 0) fail[now] = 1;
   else if(size[w] + 1 == size[q])
      fail[now] = q;
   else {
      cpy = next[w][car] = ++SAsize;
      cnt[cpy] = 0, is_clone[cpy] = true;
      firstpos[cpy] = firstpos[q];
      FOR(i,ALFA) next[cpy][i] = next[q][i];
      size[cpy] = size[w]+1;
      w = fail[w];
      fail[cpy] = fail[q];
      fail[now] = fail[q] = cpy;
      while(w && !(size[w]+1 == size[next[w][car]]))
         next[w][car]= cpy,w =fail[w];
   }
   last = now;
}
```

```
void allOcurrences(int v){
   if (!is_clone[v])
      cout << firstpos[v] - LEN + 1 << endl;
   FORR(i,ST[v].size()-1) allOcurrences(ST[v][i]);
}

int LCS2(){
   int n=1,lcp,len;
   scanf("%s",cad);
   len = strlen(cad);
   sa_init(); FOR(i,len) sa_extend(cad[i]);
   for(int i=2;i<=SAsize;i++)
      ans[i][0] = size[i];
   TopologicalSort();//Stree, top[to] es root
   while( scanf("%s",cad) != EOF){
      len = strlen(cad),lcp = 0;
      int v = root;
      FOR(i,len){
         int c = CONV(cad[i]);
         while(v && !next[v][c]) v = fail[v];
         if(v){
            lcp= min(lcp,size[v]) + 1;
            v = next[v][c];
            ans[v][n] = max(lcp,ans[v][n]);
         } else lcp = 0,v = root;
      }
      FOR(i,to){
         int v = top[i], p = fail[v];
         ans[p][n] = max(ans[p][n], ans[v][n]);
      }
      n++;
   }
   int sol = 0;
   for(int i=1;i<=SAsize;i++){
      lcp = *min_element(ans[i],ans[i]+n);
      sol = max(sol,lcp);
   }
   return sol;}
```

Hungarian

```
ll M[16][16]; int n;//minimizar- M[i][j]<0
ll Hungarian(){
   int p,q; ll xx,yy;
   vector<ll> fx(n,oo),fy(n,0);
   vector<int> x(n,-1),y(n,-1);
   for(int i=0;i<n;){
      vector<int> t(n,-1),s(n+1,i);
      for(p=q=0;p<=q && x[i]<0;++p)
         for(int k=s[p],j=0;j<n && x[i]<0;++j)
            if(fx[k]+fy[j]== M[k][j] && t[j]<0){
               s[++q]=y[j],t[j]=k;
               if(s[q]<0)
                  for(p=j;p>=0;j=p)
                     y[j]=k =t[j], p=x[k],x[k]=j;
            }
      if(x[i]<0){
         yy = oo;
         FOR(k,q+1) FOR(j,n)
            if(t[j]<0)
              yy=min(yy,(fx[s[k]]+fy[j]-
                         M[s[k]][j]));
         FOR(j,n) fy[j] += (t[j]<0 ? 0:yy);
         FOR(k,q+1) fx[s[k]] -=yy;
      } else i++;
   }
   xx = 0; FOR(i,n) xx += M[i][x[i]];
   return -xx;
}
```

Dominators Tree

```
//mark[i] = true, if i is a dominator
//stack<int> vis; <- orden del DFS
// dfsnum[i] = -1, [0..n-1]

int n,r[MAXN],C[MAXN], mark[MAXN];
int low[MAXN],father[MAXN], dfsnum[MAXN];
int find(int x){
    if(C[C[x]]==C[x]) return C[x];
    int tmp = C[x];
    C[x] = find(C[x]);
    r[x] = min(r[x] , r[tmp]);
    return C[x];
}

void Compute_Dominators(){
    FOR(i,n) low[i]=dfsnum[i],C[i]=i,mark[i]= 0;
    while(!vis.empty()){
        int v, u = vis.top(); vis.pop();
        for(int i=GT[u].size()-1;i>=0;i--){
            v = GT[u][i];
            if(v == father[u]) continue;
            if(!mark[v])
                low[u] = min(low[u],low[v]);
            else{
                int lca = queryLCA(u,v);
                if (lca != u)
                    low[u] = min(low[u],low[lca]);
                find(v);
                low[u] = min(low[u],r[v]);
            }
        }
        mark[u] = true;
        C[u] = father[u], r[u] = low[u];
    }
    memset(mark,0,sizeof(mark));
    for(int i=1;i < n;i++)
        if(low[i] >= dfsnum[father[i]])
            mark[father[i]] = true;
}
```

Hungarian Extendido

```
int N,M,cx[MAXN],cy[MAXN],w[MAXN][MAXN];
// T[i][j] = cant de x para y
// w[i][j] = -w[i][j] para Minimizar
int T[MAXN][MAXN],lx[MAXN],ly[MAXN];
void Inicializar(){
   FOR(i,N){
      lx[i]= -oo; FOR(j,M)
         T[i][j]=0, lx[i]=max(lx[i],w[i][j]);
   }
   FOR(i,M) ly[i]= 0;
}
int HungarianExt(){ // 1
   Inicializar();
   int delta,f,j,S[MAXN],SX[MAXN],P[MAXN];
   bool found,vx[MAXN],vy[MAXN];
   FOR(u,N) while(cx[u]){ // 2
      FOR(i,N) vx[i]=0, P[i] = -1;
      FOR(i,M)
         vy[i]=0,SX[i]=u, S[i]= lx[u]+ly[i]-w[u][i];
      while((vx[u] = 1)){ // 4
         delta = oo,found = 0;
         FOR(i,M) if(!vy[i]){ // 5
            delta = min(S[i],delta);
            if(S[i] == 0){ // 6
               vy[i] = 1; if(cy[i]){ // 7
                  f = min(cx[u],cy[i]);
                  for(j=SX[i];P[j]!=-1;j =SX[P[j]])
                     f= min(f, T[j][P[j]]);
                  cx[u] -=f, cy[i] -=f;
                  j= i; while(j!=-1){ // 8
                     T[SX[j]][j] += f;
                     if(P[SX[j]] != -1)
                        T[SX[j]][P[SX[j]]] -=f;
                     j= P[SX[j]];
                  } // 8
                  found = 1;
               }else  // 7
                 FOR(j,N)
                   if(!vx[j] && T[j][i]){ // 9
                      P[j]=i,vx[j]= 1;
                      FOR(k,M) if (!vy[k])
                       if(S[k]>lx[j]+ly[k]-w[j][k]){
                          S[k]= lx[j]+ly[k]-w[j][k];
                          SX[k]= j;
                       }
                   } // 9
            break;}  } // 6 5
         if(found) break;
         if(delta){
            FOR(i,N) if(vx[i]) lx[i] -=delta;
            FOR(i,M) if(vy[i]) ly[i] +=delta;
            else S[i] -=delta;
         }
   }  } // 4 2
   delta = 0;
   FOR(i,N) FOR(j,M) delta -= T[i][j]*w[i][j];
   return delta;
}
```

Biconnected components notes

```
/*
    bcc.assign(n, -1);
      for(i = 0; i < n; i++)
            if(parents[i] == -1) visitBCC(i);
*/
void visitBCC(int v) {
   for(int i=ady[v].size()-1;i>=0;i--){
      int u = ady[v][i];
      if(parents[u] == v) {
         if(low[u] < num[v]) bcc[u] = bcc[v];
         else if(low[u]>num[v]) bcc[u] = -1;// bridge
         else bcc[u] = newIndex++;
         visitBCC(u);
      }
   }
}
int getBCC(int u,int v){
      return bcc[(num[u]>num[v])? u:v];}
```

Tree isomorphism

```
int dist[MAXN], dist2[MAXN];
bool visited[MAXN + 1]; ull shaker[MAXN];
void dfs(int k, int v){
   visited[v] = true;
   FORR(i,graph[k][v].size()-1){
      int u = graph[k][v][i];
      if(visited[u]) continue;
      dist[u] = dist[v] + 1;
      dfs(k, u);
   }
}
pair<int,int> find_center(int k){
   memset(visited,0, sizeof(visited));
   dist[0] = 0, dfs(k, 0);
   FOR(i,2){
      int e = max_element(dist, dist + N)-dist;
      memset(visited,0, sizeof(visited));
      memcpy(dist2, dist, sizeof(dist));
      dist[e] = 0; dfs(k, e);
   }
   int diameter= *max_element(dist, dist+ N);
   pair<int,int> ret(-1,-1);
   FOR(i,N){
      bool ok = (dist[i]== diameter/2);
      ok |=     (dist2[i]==diameter/2);
      if(ok && dist[i]+dist2[i] == diameter){
         if(ret.first == -1) ret.first = i;
         else                ret.second= i;
      }
   }
   return ret;
}
ull rec(int k, int v){
   ull ret = 1; visited[v] = true; vector<ull> hs;
   FORR(i,graph[k][v].size()-1){
      int u = graph[k][v][i];
      if(!visited[u]) hs.push_back(rec(k, u));
   }
   sort(hs.begin(), hs.end());
   FOR(i, (int)hs.size())
      ret += hs[i] * shaker[i];
   return ret; }

ull calc_hash(int k){
   pair<int,int> center = find_center(k);
   int root = center.first;
   if(center.second != -1){
      root = N;
      int v = center.first, u = center.second;
      graph[k][root].push_back(v);
      graph[k][root].push_back(u);
   *find(graph[k][v].begin(),graph[k][v].end(),u)=root;
   *find(graph[k][u].begin(),graph[k][u].end(),v)=root;
   }
   memset(visited,0,sizeof(visited));
   return rec(k, root);
}
bool is_isomorhic(){
   for(int i=0; i<=N; i++) shaker[i] = rand();
   return calc_hash(0) == calc_hash(1);
}
```

Gomory-Hu tree

```
int father[MAXN], cut[MAXN][MAXN];
void Gusfield_Algorithm(){
   FOR(i,n) father[i]=0;
   FOR(i,n) FOR(j,n) cut[i][j]=oo;
   for(int i=1;i<n;i++){
      int flow = DINIC(i, father[i]);
      memset(vis,0,sizeof(vis));
      dfs(i);//vis[v] = si s llega a v, flow[i]<cap[i]
      for(int j=i+1;j<n;j++)
         if(vis[j] && father[j]==father[i])
            father[j] = i;
      cut[i][father[i]] = cut[father[i]][i] = flow;
      for(int j=0;j<i;j++)
       cut[i][j]=cut[j][i]=min(flow,cut[father[i]][j]);
   }
}
```

Chu-Lui/Edmons

```
int Chu_Liu_Edmonds(int n,int root) {
   int  v,u,sum,super,i,j,t;
   int C[MAXN],vis[MAXN],W[MAXN];
   GT[root].clear();
   for(i=0;i<n;i++) {
      sort(GT[i].begin(),GT[i].end());
      C[i] = i;
   }
   bool cycle = 1;
   while( cycle ) {
      for(v=cycle=0;v<n;v++) vis[v] = 0;
      vis[root] = -1;
      for(i=0,t=1; i<n ;i++,t++){
         v = u = C[i];
         if( vis[u] ) continue;
         while(!vis[v])
            vis[v]=t, v =C[GT[v][0].v];
         if( vis[v] != t ) continue;
         sum=0, super=v,cycle=1;
         while (vis[v] == t){
            vis[v]=t+1, sum += GT[v][0].w;
            v = C[GT[v][0].v];
         }
         for(j=0;j<n;j++) W[j]=oo;
         while(vis[v] == t+1){
            vis[v] = t;
            for(j=GT[v].size()-1; j > 0; j-- ) {
               int w = GT[v][j].w + sum - GT[v][0].w;
                  u = GT[v][j].v;
                  W[ u ] = min( W[ u ], w );
            }
            C[v] = super;
            v = C[GT[v][0].v];
         }
         GT[super].clear();
       for(j=0;j<n;j++) if(C[j]!=C[C[j]]) C[j]=C[C[j]];
         for(j=0;j<n;j++)
            if(W[j]<oo && C[j] != super)
               GT[super].PB(edge( j,W[j]));
               sort( GT[super].begin(), GT[super].end() );
         }
      }
      for(sum=i=0;i<n;i++)
         if(i!=root && C[i]==i) +=GT[i][0].w;
      return sum;
}
```

Stable marritage problem

```
stack<int> pila;
int n, indice[MAXN];
int MEN[MAXN][MAXN], WOM[MAXN][MAXN];
int matchW[MAXN], matchM[MAXN];
//MEN[i][j]= j-sima pref para el i-simo hombre
//WOM[i][j]=rank del j-simo hombre para la i-sima mujer
// WOM[i][0]= oo
void stable_marritage(){
   for(int i = 1; i <= n; i++) {
      matchW[i] = indice[i] = 0;
      pila.push(i);
   }
   while (!pila.empty()) {
      int m = pila.top(); pila.pop();
      while (1) {
         int w = MEN[m][++indice[m]];
         if (WOM[w][matchW[w]] > WOM[w][m]) {
            if (matchW[w] != 0)
               pila.push(matchW[w]);
            matchM[m] = w, matchW[w] = m;
            break;
         }
      }
   }
}
```

Edmons

```
int n, NewBase,Start, match[MAXN];// match[i]!=0
int P[MAXN], C[MAXN];
bool marcas[MAXN] , X[MAXN];
queue<int> Q;
int LCA(int u, int v) {
   memset(X,0,sizeof(X));
   while (true) {
      u = C[u], X[u] = 1;
      if (u == Start) break;
      u = P[match[u]];
   }
   while (1)
      if(X[v = C[v]]) break;
      else v = P[match[v]];
   return v;
}
void ResetTrace(int u) {
   while (C[u] != NewBase) {
      int v = match[u];
      X[C[u]] = X[C[v]] = 1;
      u = P[v];
      if(C[u] != NewBase)   P[u] = v;
   }
}
void BlossomContract(int u, int v) {
   NewBase = LCA(u, v);
   memset(X, 0, sizeof(X));
   ResetTrace(u), ResetTrace(v);
   if (C[u] != NewBase) P[u] = v;
   if (C[v] != NewBase) P[v] = u;
   for(u = 1; u <= n; u++)
      if (X[C[u]]) {
         if (!marcas[u]) Q.push(u);
         marcas[u] = (C[u] = NewBase);
      }
}
int FindPath() {
   while(!Q.empty()) Q.pop();
   for(int u=1; u<=n;u++)
      C[u]=u, P[u]=marcas[u]=0;
   marcas[Start] = 1; Q.push(Start);
   while (!Q.empty()) {
      int u = Q.front(); Q.pop();
      for(int i=last[u];i ;i=edges[i].next){
         int v =edges[i].v;
         if(C[u] !=C[v]&& match[u] != v){
         if(v==Start ||(match[v] && P[match[v]]))
            BlossomContract(u, v);
         else if (!P[v]) {
            P[v] = u;
            if (!match[v]) return v;
            marcas[ match[v] ] = 1;
            Q.push(match[v]);
            }
         }
      }
   }
   return 0;
}
void Aumentar(int u) {
   int v, w; while (u) {
      v = P[u], w = match[v];
      match[v] = u,match[u] = v;
      u = w;
   }
}
void Edmonds() {
   memset(match, 0, sizeof(match));
   for(Start= 1;Start<= n;Start++)
      if (!match[Start])
         Aumentar( FindPath() );
}
```

Stoer-Wagner

```
int minimumCut() {
   for(int i=0;i<n;i++) V[i] = i;
   int cut=oo;
   for(int w,v,u,m = n; m>1; m--) {
      vector<int> ws(m, 0);
      for(int i=0;i<m;i++){
    u=v;v=max_element(ws.begin(),ws.end())-ws.begin();
         w = ws[v], ws[v] = -1;
         for(int j=0;j<m;j++) if (ws[j] >= 0)
            ws[j] += cap[V[v]][V[j]];
      }
      for(int i=0;i<m;i++){
         cap[V[i]][V[u]] += cap[V[i]][V[v]];
         cap[V[u]][V[i]] += cap[V[v]][V[i]];
      }
      swap(V[v],V[m-1]); cut = min(cut, w);
   }
   return cut;
}
```

Chinese Postman

```
int chinesePostman() {
   int total = 0;
   vector<int> odds;
   for(int u=0;u<V;u++){
      for(int i=ady[u].size()-1;i>=0;i--)
         total += ady[u][i].p;
      if (ady[u].size() & 1) odds.push_back(u);
   }
   total /= 2;
   int n = odds.size(), N = 1 << n;
   int w[n][n] = FLOYD(odds), best[N];
   for(int i=0;i<N;i++) best[i] = 1e8;
   best[0] = 0;
   FOR(S,N) FOR(i,n) if (!(S & (1<<i)))
         FAB(j,i+1,n) if (!(S & (1<<j)))
   best[S|(1<<i)|(1<<j)]=min(now, best[S]+w[i][j]);
   return total + best[N-1];
}
```

Kth Shortest Path

```
struct HeapNode{
    Edge* edge;   int depth;
    HeapNode* ch[4];
    bool operator<(HeapNode* node2){
        return edge->w > node2->edge->w;
    }
} *null, *tree[MAXN];
#define CS edge->w
typedef pair<int, pair<int, Edge*> > DQI1;
typedef pair<ll, HeapNode*> DQI2;
HeapNode* createHeap(HeapNode* cur, HeapNode* now){
    if (cur == null) return now;
    HeapNode *root = new HeapNode;
    memcpy(root, cur, sizeof(HeapNode));
    if (now->edge->w < cur->edge->w){
        root->edge = now->edge;
        root->ch[2] = now->ch[2];
        root->ch[3] = now->ch[3];
        now->edge = cur->edge;
        now->ch[2] = cur->ch[2];
        now->ch[3] = cur->ch[3];
    }
    if (root->ch[0]->depth < root->ch[1]->depth)
        root->ch[0] = createHeap(root->ch[0], now);
    else
        root->ch[1] = createHeap(root->ch[1], now);
        root->depth=max(root->ch[0]->depth,
                        root->ch[1]->depth)+1;
    return root;
}
int dist[MAXN];   Edge* prev[MAXN];
vector<Edge*> graph[MAXN],graphR[MAXN];
void add_edge(int v,int u,int w){
    Edge* now = new Edge;
    now->from = v, now->to = u;
    now->w = w;
    graph[v].push_back(now);
    graphR[u].push_back(now);
}
```

```
void solve(int s,int t,int k){
    int v,d;       //Dijkstra
    queue<int> dfsOrder;
    memset(dist, -1, sizeof(dist));
   priority_queue<DQI1,vector<DQI1>,greater<DQI1> > dq;
    dq.push(MP(0, MP(t, (Edge*) NULL)));
    while (!dq.empty()){
        d = dq.top().first;
        v = dq.top().second.first;
        Edge* edge = dq.top().second.second;
        dq.pop();
        if (dist[v] != -1) continue;
        dist[v] = d, prev[v] = edge;
        dfsOrder.push(v);
        for(int i=graphR[v].size()-1;i>=0;i--){
            Edge *it = graphR[v][i];
            dq.push(MP(d + it->w, MP(it->from, it)));
        }
    }
    null = new HeapNode;
    null->depth = 0;
    null->edge = new Edge;
    null->CS = oo;
    fill(null->ch, null->ch + 4, null);
    while (!dfsOrder.empty()){
        v = dfsOrder.front();
        dfsOrder.pop();
        if (prev[v] == NULL) tree[v] = null;
        else tree[v] = tree[prev[v]->to];
        vector<HeapNode*> HEAP;
        for(int i=graph[v].size()-1;i>=0;i--){
            Edge *it = graph[v][i];
            if (dist[it->to] == -1) continue;
            it->w += dist[it->to] - dist[v];
            if (prev[v] != it){
                HeapNode* cur = new HeapNode;
                fill(cur->ch, cur->ch + 4, null);
                cur->depth= 1, cur->edge= it;
                HEAP.push_back(cur);
            }
        }
        if (!HEAP.empty()){
            make_heap(HEAP.begin(), HEAP.end());
            int size = HEAP.size();
            for (int p = 0; p < size; p++){
          HEAP[p]->ch[2]=2*p+1<size? HEAP[2*p+1]: null;
          HEAP[p]->ch[3]=2*p+2<size? HEAP[2*p+2]: null;
            }
            tree[v]= createHeap(tree[v], HEAP.front());
        }
    }
   priority_queue<DQI2,vector<DQI2>,greater<DQI2> > aq;
    if (dist[s] == -1) printf("NO\n");
    else{
        printf("%d\n", dist[s]);
        if (tree[s] != null)
            aq.push(MP(dist[s]+tree[s]->CS,tree[s]));
    }
    while (--k){
        if (aq.empty()){printf("NO\n"); continue; }
        ll d = aq.top().first;
        HeapNode* cur = aq.top().second;
        aq.pop();
        printf("%lld\n", d);
        if (tree[v = cur->edge->to] != null)
            aq.push(MP(d +tree[v]->CS,tree[v]));
        for (int i = 0; i < 4; i++)
            if (cur->ch[i] != null)
    aq.push(MP(d- cur->CS +cur->ch[i]->CS,cur->ch[i]));
    }
}
```

Treap

```
srand( time( 0 ) );   #include <time.h>
struct node {
   int key,prio, size; node *ch[2];
   node( int val = 0){
      key= val, prio = rand();
      ch[0] = ch[1] = NULL, size = 1;
   }
};
void rotate(node *&root,bool dir) {
   node *tmp = root->ch[dir];
   root->ch[dir] = tmp->ch[1-dir];
   tmp->ch[1-dir] = root;
   update_size(root); update_size(tmp);
   root = tmp;
}
void insert(node *&root,int val){
   if ( !root ) { root = new node(val); return; }
   if (val == root->key) return;
   bool dir = !( val < root->key );
   insert( root->ch[dir], val );
   if(root->prio > root->ch[dir]->prio)
      rotate( root, dir );
   update_size( root );
}
void erase(node *&root,int val){
   if (!root ) return;
   if ( val != root->key) {
      bool dir= !(val < root->key);
      erase( root->ch[dir], val );
   } else {
      node *L= root->ch[0], *R= root->ch[1];
      if (L) if(R) rotate(root, L->prio > R->prio);
      else rotate(root,0);
      else if(R) rotate(root,1);
      else { root = 0; return ; }
      erase( root, val );
   }
   update_size( root );
}
```

Kth Minimum Structure

```
struct node{
   node *l,*r; a,b,s;
} tree[MAXN*MAXLOG], *root[MAXN];
node *built(int izq,int der){
   node *q = &tree[size++];
   q->a = izq, q->b = der, q->s = 0;
   if (izq == der) return q;
   int med = (izq + der) / 2;
   q->l = built(izq,med);
   q->r = built(med+1,der);
   return q;
}
node *update(node *p,int x,int s){
   node *q = &tree[size++];
   q->a = p->a, q->b = p->b, q->s = p->s + s;
   q->l = p->l, q->r = p->r;
   if (p->a == x && p->b == x) return q;
   int med = (p->a+p->b) / 2;
   if (x <= med) q->l = update(p->l,x,s);
   else q->r = update(p->r,x,s);
   return q;
}
int FIND(node *a,node *b,int k){
   if (a->a == a->b) return a->a;
   int cant = a->l->s - b->l->s;
   if (cant >= k) return FIND(a->l,b->l,k);
      return FIND(a->r,b->r, k - cant);
}
int FIND(node *a,node *b,node *lca,int k,int vLCA){
   if (a->a == a->b) return a->a;
   int cant = a->l->s + b->l->s - 2*lca->l->s;
   if (a->l->a <= vLCA && vLCA <= a->l->b) cant++;
   if (cant >= k) return FIND(a->l,b->l,lca->l,k,vLCA);
   return FIND(a->r,b->r,lca->r, k-cant, vLCA);
}
void init(){
   root[0] = built(1,N);
   FAB(i,1,n) root[i]=update(root[i-1],arr[i],1);
}     //root[i-1] prev of root[i]
```

Fast Fourier Transform

```
Complex step[2][19];
void fft_init(){
   for(int i=1,M=2;i <= 18;i++,M <<= 1){
     step[0][i]=Complex(cos( 2*M_PI/M),sin( 2*M_PI/M));
     step[1][i]=Complex(cos(-2*M_PI/M),sin(-2*M_PI/M));
   }
}void fft(int n, Complex A[], bool paso){
   int p = __builtin_ctz(n),m,i,j,K;
   Complex a[n],w;
   for(i=0;i < n;++i) a[i] = A[i];
   for(i=0;i < n;A[i++]=a[m])
      for(m=j=0;j<p;j++) m<<=1, m|= ((i>>j)& 1);
   for(i=K=1;i <= p; i++,K<<=1)
      for(j=0; j<n ; j += K<<1 )
         for(w=1,m=j;m < K+j ;m++){
            Complex t= w * A[m + K];
            A[m + K] = A[m] - t;
            A[m]     = A[m] + t;
            w = w * step[paso][i];
         }
   if(paso) for(i=0;i<n;i++)
      A[i].x /= n, A[i].y /= n;
}

#define MAXN 262144
Complex A[MAXN],B[MAXN],C[MAXN];
void mult(int nP,ll P[],int nQ,ll Q[],int &nR,ll R[]){
   nR = nP + nQ;
   while(__builtin_popcount(nR)>1) nR += nR & -nR;
   FOR(i,nP)A[i]=P[i]; FOR(i,nQ)B[i]=Q[i];
   FAB(i,nP,nR) A[i]=0; FAB(i,nQ,nR) B[i]=0;
   fft(nR,A,0), fft(nR,B,0);
   FOR(i,nR) C[i] = A[i] * B[i];
   fft(nR,C,1);
   FOR(i,nR) R[i] = round(C[i].x);
}
```

Digit Count

```
void DigitCount(int n,ll *sol){
   ll aux=n, sum=0,p=1,d;
   while(aux){
      d = aux % 10, aux /= 10;
      sol[d] += ((n%p)+1);
      FOR(i,d)  sol[i] += p;
      FOR(i,10) sol[i] += sum*d;
      sol[0] -= p, sum= p+10*sum,p *=10;
   }
}
```

Salto del Caballo

```
ll SaltoCaballo(ll x1,ll y1,ll x2,ll y2){
   ll dx =ABS(x2-x1), dy =ABS(y2-y1);
   ll lb =max(dx+1 , dy + 1)/2;
   lb = max(lb, (dx + dy + 2)/3);
   while((lb % 2) != (dx+ dy)%2) lb++;
   if(ABS(dx)==1 && !dy) return 3;
   if(ABS(dy)==1 && !dx) return 3;
   if(ABS(dx)==2 && ABS(dy)==2) return 4;
   return lb;
}
```

Tiling Dominoes

```
double res = 1;
for(double i = 1;i<=n;i++)
   for(double j = 1;j<=k;j++){
      double x=4*cos(PI*i/(n+1))*cos(PI*i/(n+1));
      x += 4*cos(PI*j/(k+1))*cos(PI*j/(k+1));
      res *= pow(x,0.25);
}
(ll)(res+0.000001);
```

Joseph

```
int joseph (int n, int k) {
   int res = 0; FOR(i,n) res =(res+k) % (i+1);
   return res + 1;
}
```

Miller Rabin & Pollard Rho

```
ll multMOD(ll a, ll b, ll mod ){
   if( b == 0 ) return 0;
   if(b&1) return (multMOD(a,b-1,mod)+ a)%mod;
   return (2*multMOD(a,b/2, mod))%mod;
}
ll f(ll x,ll mod){
   ll rx= multMOD(x,x,mod) + 123;
   while(rx >= mod) rx -= mod;
   if(!rx) rx = 2;
   return rx;
}
bool Miller_Rabin(ll n ,ll iter){
   ll m = n-1,b=2,z; int j,a=0;
   while(!(m&1))  m>>=1, ++a;
   while(iter--){
      j =0; z = powMOD(b,m,n);
      while(!(( !j && z==1)|| z==n-1))
         if((j > 0 && z==1)|| ++j==a)
            return 0;
         else z = powMOD(z,2,n);
      b = f(b,n);
   }
   return 1;
}
bool is_prime(ll n){
   if (n == 2) return true;
   return n>1 && (n&1) && Miller_Rabin(n, 1);
}
ll factores[70]; int nfactor;
ll  pollard_rho(ll c, ll num){
   ll x=rand()%num, i=1, k=2, y=x,comDiv;
   do { i++;
      if((x =multMOD(x, x, num)-c)<0) x += num;
      if(x == y) break;
      comDiv =GCD((y-x +num) %num,num);
      if(comDiv>1 && comDiv<num ) return comDiv;
      if(i ==k) y = x, k <<= 1;
   }while ( true );
   return num;}
```

```
void fFindFactor(ll num){
   if ( is_prime(num) ){
      factores[nfactor++] = num; return; }
   ll factor = num + 1;
   while(factor >= num)
      factor= pollard_rho(rand()%(num-1)+ 1,num);
   fFindFactor(factor);
   fFindFactor(num / factor);
}
```

Fechas

```
Calendar calendar = Calendar.getInstance();
calendar.set(Calendar.YEAR , y);
calendar.set(Calendar.MONTH, m);
calendar.set(Calendar.DAY_OF_MONTH, d);
DIA = calendar.get(Calendar.DAY_OF_WEEK);
```

Compilador

```
import javax.script.*;
ScriptEngineManager manager = new
ScriptEngineManager();
ScriptEngine motor = manager.getEngineByName("js");
motor.put("VARIABLE", valor);
motor.eval(Expresion);
```

Expresiones Regulares

```
import java.util.regex.*;
Pattern pattern = Pattern.compile(expresion);
Matcher matcher = pattern.matcher(patron);
if (matcher.matches())
```

Simpson

```
double Simpson(double a,double b){
   double s = 0,h = (b - a) / ITR;
   for (int i = 0; i <= ITR; ++i) {
      double x = a + h * i;
      s += f(x)*((i==0||i ==ITR)?1: ((i&1)==0)? 2:4);
   }
   return s * h/3;
}
```

Number Theory

```
ll GCDext(ll a, ll b, ll &x, ll &y){
   ll g = a; x = 1 ; y = 0;
   if (b != 0){
      g = GCDext(b, a % b, y, x);
      y -= (a / b) * x;
   }
   return g;
}
ll invMod(ll a, ll m, ll &inv) {
   ll x, y;
   if(GCDext(a,m,x,y) != 1) return 0;//noSolucion
   inv = (x + m) % m;
   return 1;
}
void GetAllInverseElements(int mod){
   ll inv[mod]; inv[1] = 1;
   for (int i=2; i<mod; i++){
      inv[i]= (mod/i) * inv[mod % i];
      inv[i]= (mod - inv[i] %mod)% mod;
   }
}
//ax+by=c  return x,y,bool = c|gcd(a,b)
bool mExtGcd(ll a,ll b,ll c,ll &x,ll &y){
    ll r = GCDext(a,b,x,y);
    if (c % r) return false;
    x *= c/r, y *= c/r;
    return true;
}
```

```
//ax % b = m
ll modeq(ll a,ll b,ll m){
   ll x,y,d = GCDext(a,m,x,y);
   if(b % d) return -1;//no solution
   ll xi = ((x+m)%m * (b/d)%m)% m;
   /*for(ll i = 0;i < d;i++){
      xi += i * (m/d);xi%= m;
      cout<<xi<<endl;
   }*/
   return xi;
}

// x = a[i] mod m[i]
// if GCD(m[i],m[j]) != 1 -> noSolucion
bool RestoChino(int n,ll *a,ll *m,ll *x){
   ll K = 1, inverso;  *x = 0;
   FOR(i,n) K *= m[i];
   FOR(i,n){
      invMod(K/m[i],m[i],inverso);
      *x += a[i]* K/m[i]* inverso;
   }
   *x %= K;
   return 1; // Tiene sol
}
```

Index Permutation

```
int alpha[30]; // Ya Normalizado
ll IndexPermutation(int *per,int n,int dif){
   //n=len, dif -elems diferentes
   memset(alpha,0,sizeof(alpha));
   FOR(i,n) alpha[per[i]]++;
   ll sol=0; FOR(i,n-1){
      FAB(j,1,per[i]){
         if(!alpha[j]) continue;
         ll par = FACT(n-i-1);
         FAB(k,1,dif+1) par /=FACT(alpha[k]-(k==j));
        sol += par;
      }
      --alpha[per[i]];
   }
   return sol;
}
```

Baby-step giant-step

```
int h[MAXN],from[MAXN];
void Insert(int sta, ll x){
   int pos = x % MAXN;
   while(h[pos]!=-1 && h[pos] != x) pos=(pos+1)%MAXN;
   h[pos]=x, from[pos]=sta;
}
int Find(ll x){
   int pos = x % MAXN;
   while(h[pos]!=-1 && h[pos] != x) pos=(pos+1)%MAXN;
   return (h[pos] < 0)? -1 : from[pos];
}
int baby_step(ll K, ll N, ll P, int ok){
   K %= P; N %= P;
   if (!ok && N == 1) return 0;
   int M = static_cast<ll>(sqrt(P));
   memset(h, -1, sizeof h);
   ll A = N * K % P, B = K;
   FOR(i,M){
      Insert(i+1, A);
      if (ok && B == N) return i+1;
      A = A*K % P, B = B*K % P;
   }
   A = B = mod_pow(K, M, P);
   for(int i =ok+1; i*M-M <= P;i++,  B = B*A % P ){
      int x = Find(B); if(x>=0) return i*M-x;
   }
   return -1;
}
// A^x = B mod M
int Solve(ll A, ll B, ll M){
   A %= M; if (B >= M) return -1;
   ll POT = 1,aux,tN = B;
   FOR(i,41)
      if (POT == B) return i;
      else POT = POT * A % M;
   while ((aux = GCD(M, A)) != 1) {
      if (tN % aux) return -1;
      tN /= aux, M /= aux;
   }
   int ans = baby_step(A, B, M, 0);
   if (ans < 0) return -1;
   if (ans <= 40) {
      int pi = baby_step(A, 1, M, 1);
      while (ans <= 40) ans += pi;
   }
   return ans;
}
```

Kth Permutation

```
int N; // N grupos
char grupo[22];//caract del grupo
int cantgrupo[22];ll quitar;
//FOR(i,N) quitar *= fac[cantgrupo[i]]
void KthPermutacion(int k,int quedan){
   if (quedan == 0) return;
   ll total=FACT(quedan- 1),inicio=0,fin=0;
   FOR(i,N){
      if (cantgrupo[i] == 0) continue;
      fin += (total * cantgrupo[i]) / quitar;
      if (fin > k){
         quitar /= cantgrupo[i]--; cout << grupo[i];
         KthPermutacion(k-inicio,quedan-1);
      } else inicio = fin;
   }
}
```

Pick Theorem

```
A = I + B/2 - 1

int ptsSegment(point a, point b){
   int aa= abs(b.y-a.y),bb= abs(b.x-a.x);
   if(aa==0 && bb== 0)  return 0;
   if(aa==0) return bb-1;
   if(bb==0) return aa-1;
   return gcd(aa, bb) - 1;
}
ll ptsBoundary(Pol p,int n){
   ll ans=n;FOR(i,n) ans+=ptsSegment(p[i],p[(i+1)%n]);
   return ans;
}
```

Gauss 01

```
int n,mat[MAXR*MAXC+1][MAXR*MAXC+1];
int X,x[MAXR*MAXC+1],index_[MAXR*MAXC+1];
//mat[v][0] - paridad de toques
//mat[v][u] = 1, si v change u
void S(int pos, int step){
   if (step == X) return;
   if (pos == 0){ X = min(X,step); return; }
   if (!mat[index_[pos]][pos]){
      x[pos] = 1; S(pos - 1, step + 1);
      x[pos] = 0; S(pos - 1, step);
      return;
   }
   int left = 0;
   for (int j = n; j > pos; j--)
      if (mat[index_[pos]][j]) left ^= x[j];
   x[pos] = mat[index_[pos]][0]^left;
   x[pos]? S(pos - 1, step + 1) : S(pos - 1, step);
}
int G(){
   bool no = false; FOR(i,n) index_[i+1] = i+1;
   for (int i = 1; i <= n && !no; i++) {
      FAB(j,i,n+1) if (mat[index_[j]][i])
         {swap(index_[i],index_[j]); break;}
      if(mat[index_[i]][i] == 0) continue;
      FAB(j,i+1,n+1)
         if (mat[index_[j]][i]){
            FAB(k,i,n+1)
               mat[index_[j]][k] ^= mat[index_[i]][k];
            mat[index_[j]][0] ^= mat[index_[i]][0];
            if (mat[index_[j]][0]){
               bool left = 0;
               for (int k=i+1; k <= n && !left;k++)
                  left |= mat[index_[j]][k];
               if (!left){ no = true; break; }
            }
         }
   }
   if (no) return -1;//No solution
   X = n + 1; S(n, 0); return X;}
```

Resto Chino No Coprimos

```
bool RestoChinoNoCoprimos( int n,ll *a,ll *m,ll *x )  {
   ll K = 1, lo = *max_element(a,a+n);
   FOR(i,n) a[i] %= m[i];

  //para cada primo distinto,guardar la max pot usada en
  // alguna fact y a[i] mod la pot
   map< int, vector< par > > M;
   FOR(i,n) {
      int x = m[i];
      for( int d = 2; d*d <= x; ++d ){
         if (x % d) continue;
         int pot = 1;
         while (x%d== 0) x /=d, pot *=d;
         M[d].push_back(par(pot, a[i]% pot ));
      }
      if( x > 1 ) M[x].push_back( par( x, a[i] % x ) );
   }
   //Foreach M in it{
      vector< par > &v = it->second;
      sort( v.begin(), v.end() );
      par z = v.back();
      FOR(i,n) if(z.b% a[i] != m[i]) return 0;
      K *= z.a;
   }

   *x = 0;
   //Foreach M in it{
      par z = it->second.back();
      ll ai = z.b, mi = z.a, inverso , x1,y2;
      GCDext(K / mi, mi, x1, y2);
      inverso = (x1 + mi) % mi;
      *x += ai * (K/mi) * inverso;
   }
   while( *x < lo ) *x += K;
   return 1;
}
```

Simplex

```
// FO:  (min)z = X1*T[0][1]+ .. Xn*T[0][n]
// SA: X1*T[1][1]+X2*T[1][2]... <= val[1]
//     X1*T[m][1]+X2*T[m][2]... <= val[m]

int n;  // variables - tipos de obj
int m;  // restricciones - concursantes
double T[REST][REST+VAR],val[REST];

void READ(){
   scanf("%d%d",&n,&m);
   FOR(j,m+1) FOR(i,m+n+1) T[j][i] = 0.00;
   // Objective fct
   FAB(i,1,n+1) scanf("%lf", &T[0][i]);
   FAB(i,1,n+1) T[0][i] = -T[0][i];//Max *= -1
   val[0] = 0;
   // m constraints
   FAB(row,1,m+1){
      FAB(i,1,n+1) scanf("%lf", &T[row][i]);
      scanf("%lf", &val[row]);
   }
}
double Simplex(){
   T[0][0] = 1; FAB(row,1,m+1) T[row][n+row] = 1;
   while(true){
      double min = 0.000; // Find entering variable
      int enter_col = -1;
      FOR(i,n+m+1) if(T[0][i] < min-EPS)
         min = T[0][i], enter_col = i;
      // Check if iteration ends
      if(enter_col == -1 || fabs(min)<EPS)
         break;//      puts("Iteration ends");
      // Find entering variables
      double mrt = -1; int mrt_row = -1;
      for(int row=1; row<=m; row++){
         if(fabs(T[row][enter_col]) < EPS) continue;
         double tmp = val[row] / T[row][enter_col];
         if(tmp >= EPS || fabs(tmp)< EPS)
         if( mrt<0 || tmp<mrt) mrt=tmp,mrt_row = row;
      }
      /*  mrt_row == -1 -> Solución ilimitada */
      // Gaussian Elimination for other rows
      for(int row=0; row<=m; row++){
         if(row == mrt_row) continue;
         if(fabs(T[row][enter_col]) < EPS) continue;
double factor=-T[row][enter_col]/T[mrt_row][enter_col];
         T[row][enter_col] = 0.0000;
         for(int col=1; col<=n+m; col++)
            if(col == enter_col) continue;
            else T[row][col]+=factor*T[mrt_row][col];
         val[row] += factor*val[mrt_row];
      }
      // Normalize row - mrt_row
      for(int col = 1; col <= n+m; col++){
         if(col == enter_col) continue;
         T[mrt_row][col] /= T[mrt_row][enter_col];
      }
      val[mrt_row] /= T[mrt_row][enter_col];
      T[mrt_row][enter_col] = 1;
   }
   return val[0];
}

//Minimal Enclosing Circle
void min_circle(P p[],int n,P &c,double &r){
   random_shuffle(p,p+n); c = p[0]; r = 0;
   FAB(i,1,n) if(abs(p[i]-c) > r+EPS){
      c = p[i], r = 0;
      FOR(k,i) if( abs(p[k]-c) > r+EPS ){
         c= P((p[i].X+p[k].X)/2, (p[i].Y+p[k].Y)/2);
         r = abs(p[k]-c);
         FOR(j,k) if(abs(p[j]-c) > r+EPS){
            c = circunferenceCenter(p[i],p[k],p[j]);
            r = abs(p[i]-c);
         }
      }
   }
}
```

Maximal Independent Set

```
int best,valor[MAXN];
bool ady[MAXN][MAXN];
void solve(VI oldSet,int ne,int N,int curSum){
   int nod = 0,minnod = N,fixp= -1,s= -1;
   for(int i=0; i<N && minnod != 0; i++){
      int p= oldSet[i],cnt= 0,pos= -1;
      FAB(j,ne,N) if (ady[p][oldSet[j]]) {
            if (++cnt == minnod) break;
            else pos = j;
         }
      if (minnod > cnt) {
         minnod = cnt, fixp = p;
         if (i < ne) s = pos; else s = i, nod = 1;
      }
   }
   VI newSet(N);
   for(int k = minnod + nod; k >= 1; k--) {
      int sel = oldSet[s], newne=0;
      swap(oldSet[s], oldSet[ne]);
      FOR(i,ne) if (!ady[sel][oldSet[i]])
         newSet[newne++] = oldSet[i];
         int newce = newne; long remain = 0;
         FAB(i,ne+1,N)
            if (!ady[sel][oldSet[i]]) {
               newSet[newce++] = oldSet[i];
               remain += valor[oldSet[i]];
            }
      curSum += valor[sel];
      if (newce == 0){
         best = max(best, curSum); // ready
      }else if (newne < newce) {
         if (curSum + remain > best)
            solve(newSet, newne, newce, curSum);
      }
      curSum -= valor[sel];
      ++ne; if (k > 1)
         {s = ne;while(!ady[fixp][oldSet[s]]) s++;}
   } }
```

```
int maximumIndependentSet(int N){
   VI all(N); FOR(i,N) all[i] = i;
   best = 0;   solve(all, 0, N, 0);
   return best;
}
```

Link-Cut notes

```
-hacer el EXPOSE en su función básica,recordar hacer
splay luego.
-en el EVERT no fijar el 1, hacer XOR, hacerlo raíz y
p->rev ^= 1, mas nada;
-para el LINK hacer un EVERT
-para un camino, los datos están en el nodo individual,
el derecho que se arrastra y el derecho del nodo.
-hacer la prop implementando un push_down en splay.
-en push_down subir hasta que sea is_root().
```

```
void Cut(node *x,node *y){
   expose(y), splay(x);
   if(x->p == y) x->p = 0;
    else
      expose(x), splay(y), y->p = 0;
}
bool same_root(node *p,node *q){
   expose(p),splay(p);
   expose(q),splay(q);
   return p->p != NULL;
}
```

Splay Trees notes

```
-siempre implementar el find, y prop mientra bajo.
-luego de cualquier OP en un intervalo hacerle UPDATE a
los 2 padres, root y root->l o r.
-implementar el BUILT.
```

Geometria Computacional

```
const double PI = 3.141592653589793;
#define X real()
#define Y imag()
typedef complex<double> P;
typedef vector<P> Pol;
struct circle{
   P p; double r;
   circle(P p=0,double r=0):p(p),r(r){}
};
struct L: public vector <P> //Linea
   {L(P a,P b){PB(a); PB(b);}};

bool cmp(const P a, const P b)
   {return a.X!=b.X ?a.X<b.X :a.Y <b.Y;}

  //<-   .........   ->\\

double cross(P a, P b)//1
   {return a.X*b.Y-a.Y*b.X;}
double dot(P a, P b)//2
   {return a.X*b.X + a.Y*b.Y;}

//Orientacion de 3 puntos
int ccw(P a, P b, P c){ //3,1 2
   double d = cross(b-a,c-a);
   if(d > EPS) return +1;
   if(d < EPS) return -1;
   return 0;
}
//Interseccion de 2 rectas
bool intersectLL (L l, L m){//4,1
   //non-parallel
   return abs(cross(l[1]-l[0], m[1]-m[0])) > EPS
       || abs(cross(l[1]-l[0], m[0]-l[0])) < EPS;
} //same-line
```

```
//Punto interseccion recta recta
P crosspoint(L l, L m){ //5,1
   double A = cross( l[1]-l[0], m[1]-m[0]);
   double B = cross( l[1]-l[0], l[1]-m[0]);
   if(abs(A)<EPS && abs(B)<EPS)
      return m[0]; //Same line
   if(abs(A)<EPS) return P(0,0);//parallels
   return m[0] + B / A * (m[1] - m[0]);
}
//Interseccion recta y segmento
bool intersectLS (L l, L s){//6, 1
   //s[0] is left of l
   return cross(l[1]-l[0], s[0]-l[0]) *
          cross(l[1]-l[0], s[1]-l[0]) < EPS;
} //s[1] is right of l

//Interseccion recta y punto
bool intersectLP (L l, P p)//7,1
   {return abs(cross(l[1]-p, l[0]-p))<EPS;}

//Interseccion de 2 segmento
bool intersectSS (L s, L t){//8,3
   FOR(i,2)FOR(j,2) if(abs(s[i]-t[j])<EPS)
      return 1;  // same point
  return ccw(s[0],s[1],t[0])*ccw(s[0],s[1],t[1]) <= EPS
      && ccw(t[0],t[1],s[0])*ccw(t[0],t[1],s[1]) <= EPS;
}
//Interseccion segmento y punto
bool intersectSP (L s,P p){//9
   double a=abs(s[0] - p) + abs(s[1] - p);
   return a - abs(s[1]-s[0]) < EPS;
}
//Proyeccion punto recta
P projection(L l,P p){//10,2
  double t=dot(p-l[0], l[0]-l[1])/norm(l[0]-l[1]);
  return l[0] + t*(l[0]-l[1]);
}
//Refleccion punto recta
P reflection(L l, P p)//11, 10
   {return p +(P(2,0) *(projection(l,p)-p));}
```

```
//Distancia recta punto
double distanceLP(L l,P p)//12, 10
   {return abs(p - projection(l,p));}
//Distancia recta recta
double distanceLL(L a, L b){//13,4 12
   if(intersectLL(a,b)) return 0;
   return distanceLP(a,b[0]);
}
//Distancia recta segmento
double distanceLS(L l, L s){//14,7 12
  if(intersectLS(l,s)) return 0;
  return min(distanceLP(l,s[0]),distanceLP(l,s[1]));
}
//Distancia segmento punto
double distanceSP(L s, P p){//15, 10 9
   const P r = projection(s,p);
   if (intersectSP(s,r)) return abs(r-p);
   return min( abs(s[0]-p), abs(s[1]-p) );
}
//distancia segmento segmento
double distanceSS (L s, L t) {//16,8 15
   if (intersectSS(s, t)) return 0;
   double a=oo;FOR(i,2)a=min(a,distanceSP(s,t[i]));
   double b=oo;FOR(i,2)b=min(b,distanceSP(t,s[i]));
   return min(a,b);
}
//Centro de circunferencia dado 3 puntos
P circunferenceCenter(P a, P b, P c){//17
   P x =1.0/conj(b-a), y=1.0/conj(c-a);
   return (y-x)/(conj(x)*y-x*conj(y)) +a;
}
P ComputeCentroid(const Pol &pol) {
   P c(0,0);int n=pol.size();
   double scale = 6.0 * area(pol);
   FOR(i,n){
      int j = (i+1)%n;
      c += (pol[i] + pol[j])*cross(pol[i],pol[j]);
   }
   return c / scale;
}
```

```
// arg(a)
double anguloEjeX(P a){//18,1 2
   P b = P(1,0);
   if(dot(b,a)/(abs(a)*abs(b))==1)  return 0;
   if(dot(b,a)/(abs(a)*abs(b))==-1) return PI;
   double aux=asin(cross(b,a)/(abs(a)*abs(b)));
   if(a.X < -EPS && a.Y > EPS) aux+=PI/2;
   if(a.X < -EPS && a.Y > EPS) aux-=PI/2;
   if(aux < 0) aux += 2*PI;
   return aux;
}
double anguloEntreVectores(P a, P b)//19
      {return acos(dot(a,b)/abs(a)/abs(b));}
double anguloEntre3Puntos(P a, P b, P c)//20,19
   {return anguloEntreVectores(a-b,c-b);}

P RotarPunto(P p,double ang){
   double x = p.X*cos(ang)-p.Y*sin(ang);
   double y = p.X*sin(ang)+p.Y*cos(ang);
   return P(x,y);
}

double area(Pol pol) {//25, 1
   double A = 0; int n = pol.size();
   FOR(i,n) A += cross(pol[i],pol[(i+1)%n]);
   return A / 2;
}

Pol convexHull(Pol ps){//21,3
  int t,i,n = ps.size(), k=0;
  if (n < 3) return ps;
  sort(ps.begin(), ps.end(), cmp);
  Pol ch (2*n);
  for(i=0;i<n;ch[k++]=ps[i++]) //lower
    while(k>=2 && ccw(ch[k-2],ch[k-1],ps[i]) <= 0) --k;
  for(i=n-2,t=k+1 ;i>=0; ch[k++]=ps[i--])// upper
     while(k>=t && ccw(ch[k-2],ch[k-1], ps[i])<=0) --k;
  ch.resize(k-1);
  return ch;
}
```

```
int pointInPolygon(Pol pol, P p){//22, 1 2
   bool in = false; int n=pol.size();
   FOR(i,n){
      P a= pol[i] - p, b= pol[(i+1)%n]-p;
      if(a.Y > b.Y) swap(a,b);
      if(a.Y<=0 && 0 < b.Y)
          if (cross(a,b) < 0) in ^= 1;
      if(abs(cross(a,b)) <= EPS &&dot(a,b) <= 0)
         return true; // ON
   }
   return in; // IN | OUT
}
// anticlockwise, left of the line
Pol convex_cut(Pol pol,L l) {
   Pol Q; int n = pol.size();
   for (int i = 0; i < n; i++) {
      P A = pol[i], B = pol[(i+1)%n];
      if(ccw(l[0],l[1],A) != -1) Q.push_back(A);
      if(ccw(l[0],l[1],A)*ccw(l[0],l[1],B)<0)
      Q.push_back(crosspoint(L(A, B), l));
   }
   return Q;
}

pair <P,P> closestPair (Pol p) {//23
   int i,n = p.size(), s=0, t=1, m=2;
   vector<int> S(n);  S[0]=0, S[1]=1;
   sort(p.begin(), p.end(), cmp);
   double d = norm(p[s]-p[t]);
   for(i =2;i<n; S[m++] = i++)
      FOR(j,m){
         if(norm(p[S[j]]-p[i])<d)
            d = norm(p[s = S[j]] - p[t = i]);
         if(p[S[j]].X < p[i].X-d)
            S[j--] = S[--m];
      }
   return make_pair( p[s], p[t] );
}


//max distance pair points, O(n)
double diameter(Pol pt) {//24, 1
   int is=0,js=0, n=pt.size();
   FAB(i,1,n){
      if(pt[i].Y >pt[is].Y) is = i;
      if(pt[i].Y <pt[js].Y) js = i;
   }
   double maxd=norm(pt[is] - pt[js]);
   int i,maxi,j,maxj;
   i = maxi = is;  j = maxj = js;
   do {
   if(cross(pt[(i+1)%n]-pt[i], pt[(j+1)%n]-pt[j]) >= 0)
       j=(j+1)%n; else i=(i+1)%n;
     if (norm(pt[i]-pt[j]) > maxd){
         maxd =norm(pt[i]-pt[j]);
         maxi = i, maxj = j;
     }
   }while(i!=is || j!=js);
   return maxd;
}
//Interseccion circulo circulo
pair<P, P> intersectCC(circle a,circle b) {
   P x= b.p - a.p;
   P A= conj(x), C = a.r*a.r*(x);
   P B= (b.r*b.r - a.r*a.r -x*conj(x));
   P D= B*B-4.0*A*C;
   P z1= (-B+sqrt(D)) / (2.0*A) +a.p;
   P z2= (-B-sqrt(D)) / (2.0*A) +a.p;
   return pair<P, P>(z1, z2);
}
//Interseccion circulo linea
vector<P> CircleLineIntersection(P a,P b,P c,double r){
   vector<P> ret; b -=a, a -=c;
   double A= dot(b,b), B= dot(a,b);
   double C= dot(a,a)-r*r, D= B*B - A*C;
   if (D < -EPS) return ret;
   ret.push_back(c+a+b*(-B+sqrt(D+EPS))/A);
   if (D > EPS) ret.push_back(c+a+b*(-B-sqrt(D))/A);
   return ret;
}
```

Convex Hull 3D

```
struct Facet {int a, b, c;};
Point  operator-(Point a, Point b)
   {return Point(a.x-b.x, a.y - b.y, a.z - b.z);}

Point  operator*(Point a,Point b)
   {return Point(a.y*b.z-a.z*b.y,a.z*b.x-a.x*b.z,
                 a.x*b.y-a.y*b.x);}
double operator^(Point a,Point b)
   {return a.x*b.x + a.y*b.y + a.z*b.z;}


Point Pol[MAXN];
int vis[MAXN][MAXN];
bool iszero(double x){return fabs(x)<EPS;}
bool iszero(Point a)
   {return iszero(a.x) && iszero(a.y) && iszero(a.z);}
bool coface(Point a,Point b)
   {return (a^b)>0&& iszero(a*b);}
Point normal(Facet& f)
   {return (Pol[f.b]-Pol[f.a])*(Pol[f.c]-Pol[f.a]);}

vector<Facet> convex_hull(int N){
   int i; vector<Facet> cur;
   for (i=1; i<N; ++i)
      if (!iszero(Pol[0]-Pol[i])) {
         swap(Pol[1], Pol[i]);break;}
   if (i == N) return cur;
   for (++i;i<N;++i)
      if(!iszero((Pol[1]-Pol[0]) * (Pol[i]-Pol[0]))){
         swap(Pol[2], Pol[i]);break;}
   if (i == N) return cur;

   Point n = (Pol[0]-Pol[1])*(Pol[1]-Pol[2]);
   for (++i; i<N; ++i)
      if (!iszero(n ^ (Pol[0] - Pol[i])))
         {swap(Pol[3], Pol[i]);break;}
   if (i == N) return cur;
   cur.PB((Facet){0,1,2});
   cur.PB((Facet){2,1,0});
   for (int i=3; i<N; ++i){
      vector<Facet> next;
      for (int j=0; j<(int)cur.size(); ++j){
         Facet& f = cur[j];
         double d = (Pol[f.a] - Pol[i]) ^ normal(f);
         if (d >= 0) next.PB(f);
         int side = 0;
         if (d > 0) side = +1;
         if (d < 0) side = -1;
         vis[f.a][f.b] = side;
         vis[f.b][f.c] = side;
         vis[f.c][f.a] = side;
      }
      for (int j=0; j<(int)cur.size(); ++j){
         Facet& f = cur[j];
         int a = f.a, b = f.b, c = f.c;
         if (vis[a][b] < 0 && vis[a][b] != vis[b][a])
            next.PB((Facet){a, b, i});
         if (vis[b][c] < 0 && vis[b][c] != vis[c][b])
            next.PB((Facet){b, c, i});
         if (vis[c][a] < 0 && vis[c][a] != vis[a][c])
            next.PB((Facet){c, a, i});
      }
      cur = next;
   }
   return cur;
}
vector<Facet> tri;
int triples = tri.size(); // luego
// AREA
double dist(Point a){return sqrt(a^a);}
double area(Point a,Point b,Point c)
   {return dist((b-a)*(c-a));}
double area(){
   double ret=0;
   for(int i=0;i < triples;i++)
ret += area(Pol[tri[i].a],Pol[tri[i].b],Pol[tri[i].c]);
  return ret * 0.5;
}
```

```
// VOLUMEN
double vol(Point a,Point b,Point c,Point d)
   {return (b-a)*(c-a)^(d-a);}
double vol(){
   Point p(0,0,0); double ret = 0;
   for(int i=0;i < triples;i++)
ret+=vol(p,Pol[tri[i].a],Pol[tri[i].b],Pol[tri[i].c]);
   return fabs(ret / 6);
}

//Face
int facetri() {return triples;}
bool same(int s,int e){
   Point
a=Pol[tri[s].a],b=Pol[tri[s].b],c=Pol[tri[s].c];
   return iszero(vol(a,b,c,Pol[tri[e].a]))
       && iszero(vol(a,b,c,Pol[tri[e].b]))
       && iszero(vol(a,b,c,Pol[tri[e].c]));
}
int facepolygon(){
   int ans=0,i,j,k;
   for(i=0;i < triples;i++, ans += k)
      for(j=0,k=1;j<i;j++)
         if(same(i,j)) k=0, j=i;
   return ans;
}
```

Extreme Points

```
#define up(u,v)          (dot(u,v) > 0)
#define down(u,v)        (dot(u,v) < 0)
#define dr(u,Vi,Vj)      (dot(u, (Vi)-(Vj)))
#define above(u,Vi,Vj)   (dr(u,Vi,Vj) > 0)
#define below(u,Vi,Vj)   (dr(u,Vi,Vj) < 0)
int maximun(P U, Pol ch, int n){
   P A = ch[1] - ch[0];
   int a=0, b=n, c, upA = up(U,A), upC;
   if (!upA && !above(U, ch[n-1], ch[0])) return 0;
   while(true) {
      c = (a + b) / 2;
      P C = ch[c+1] - ch[c];
      upC = up(U,C);
      if (!upC && !above(U,ch[c-1],ch[c]))
         return c;
      if (upA) {
         if (!upC || above(U,ch[a],ch[c]))
            b = c;
         else a = c, A = C, upA = upC;
      }
      else {
         if (!upC && below(U,ch[a],ch[c]))
            b = c;
         else a = c, A = C, upA = upC;
      }
      if (b <= a+1) return -1;
   }
   return -1;
}
```

```
1+x^2+ x^3 +...+ x^k = (x^(k+1)-1 )/(x-1)
M : cantidad de Aristas
N : # de vértices
P : # de componentes conexas.
NC =  M - N + P  cantidad de ciclos.
```

Digitos de N!, n > 3

```
   0.5*log10(2*n*PI)+n*log10(n/M_E)+1;
```

Complex

```
 p1*p2->(x1*x2-y1*y2  ,x1*y2+y1*x2 )
 p1/p2 ->(x1*x2+y1*y2 ,y1*x2-x1*y2)
   cada comp / dot(p2,p2)
```

Catalan

```
   C[n] => FOR(k=0,n-1) C[k] * C[n-1-k]
   C[n] => Comb(2*n,n) / (n + 1)
   C[n] => 2*(2*n-3)/n * C[n-1]
```

Fibonacci

```
- Sumatoria de F[1..n]=F[n+2]-1.
- Si n es divisible por m => Fn es divisible por Fm
-Si N es Fibonacci=>(5*N*N+4 || 5*N*N-4) es un cuadrado
-Suma de n terminos partiendo del k-simo + k = F[k+n+1]
- gcd(F[p], F[n]) = F[gcd(p,n)] = F[1] = 1
- Cantidad num fibonacci hasta n
  floor((log10(n)+ (log10(5)/2))/log10(1.6180));
```

Girar Grilla 45 grados

```
   r = (max(col, filas) << 1) + 10;
   c = (max(col, filas) << 1) + 10;
   xx = x + y + 5;
   yy = x - y + filas + 5;
```

Cant de Palindromes de <= N Digitos

```
   a(n) = 2 *(10^(n/2) -1) si n es par
   a(n) = 11*(10^(n-1)/2)-2 si n es impar
```

Teoria de Numeros

```
   N = p^a*q^b*r^c
   CantDiv = D = (a+1)*(b+1)*(c+1)
   SumaDiv = FOR(i,k)
      Sum *= (prim[i]^(cant[i]+1)-1)/(prim[i]-1)
   ProdDiv = P = N^(D/2) = Sqrt(N^D)
```

All submask of a mask

```
for (int m=0; m<(1<<n); ++m)
   for (int s=m; s; s=(s-1)&m)
```

Max Flow Min Cost

```
bool Disjtra(){
    FOR(i,n) D[i]=oo, marcas[i]=0;
    D[s] = 0, marcas[s] = 1;
    while(!cola.empty()){
        int v = cola.front();cola.pop(); marcas[v] = 0;
        for(int i = last[v]; i ; i = edge[i].next){
            int u = edge[i].v;
            if(edge[i].cap && D[u]>D[v]+edge[i].cost){
                prev[u]= i, D[u]= D[v]+edge[i].cost;
                if(!marcas[u])
                    marcas[u]=1, cola.push(u);
            }
        }
    }
    return D[t] != oo;
}
pair<int,int> MCMF(){
   int flow = 0, cost = 0, now,u,p;
   while(Disjtra()){
      int f = oo;
      for(u = t; u != s; u = edge[p^1].v)
         p = prev[u], f = min(f, edge[p].cap);
      now = 0;
      for(u = t; u != s; u = edge[p^1].v){
         p = prev[u];
         edge[p].cap -= f, edge[p^1].cap += f;
         now += edge[p].cost;
      }
      cost += f * now, flow += f;
   }
   return MP(flow,cost);
}
```

Teorema de las medianas

$$c^2 = \frac{8(Ma)^2 + 8(Mb)^2 - 4(Mc)^2}{9}$$

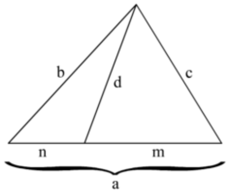
$$b^2 = \frac{6c^2 - 8(Ma)^2 - 4(Mb)^2}{-3}$$

$$a^2 = 2(b^2 + c^2) - 4(Ma)^2$$

$$a^2 = \frac{b^2}{2} - c^2 + 2(Mb)^2$$

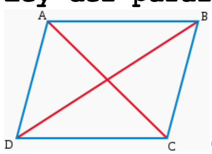
$$a^2 = -b^2 + \frac{c^2}{2} + 2(Mc)^2$$

Teorema de Stewart



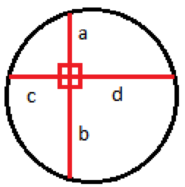
$$d^2a = nc^2 + mb^2 - nma$$

Ley del paralelogramo



$$(AB)^2 + (BC)^2 + (CD)^2 + (DA)^2 = (AC)^2 + (BD)^2$$

Cuerdas perpendiculares



$$a^2 + b^2 + c^2 + d^2 = diámetro^2$$

a -> apotema

r -> radio

c -> cuerda

$c = 2((r^2 - a^2)^{0.5})$

$c = 2r(\sin(\frac{\alpha}{2}))$

Incentro(Bisectrices)

(Xa,Ya)(Xb,Yb)(Xc,Yc) coordenadas de Pa,Pb,Pc

Coordenadas del incentro: $(\frac{aXa + bXb + cXc}{a+b+c}, \frac{aYa + bYb + cYc}{a+b+c})$

a,b,c lados opuestos a Pa,Pb,Pc

Baricentro(Medianas) $(\frac{X1+X2+X3}{3}, \frac{Y1+Y2+Y3}{3})$

El volumen de un sólido generado por el giro de un área comprendida entre dos gráficas, f(x) y g(x) definidas en un intervalo [a,b] alrededor de un eje

***Rotación paralela al eje de abscisas (Eje x: y = K)**

$$V = \pi \int_a^b \left([f(x) - K]^2 - [g(x) - K]^2\right) dx$$

***Rotación paralela al eje de ordenadas (Eje y: x = K)**

$$V = 2\pi \int_a^b (x - k)[f(x) - g(x)]\, dx$$

Superficie

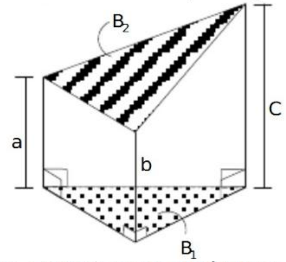
$$S = \int_a^b 2\pi f(x)\sqrt{1 + [f'(x)]^2}\, dx$$

Cubo: $V = \frac{(d^3 \sqrt{3})}{9}$

1. tronco de prisma triangular recto

$S_L = \sum \text{área de caras laterales}$

$S_T = S_L + B_1 + B_2 \quad V = B_1\left(\frac{a+b+c}{3}\right)$



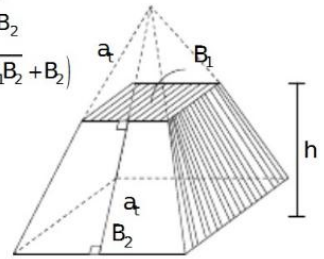
III. TRONCO DE PIRÁMIDE

1. Tronco de Pirâmide regular

$S_L = \left(P_{B_1} + P_{B_2}\right)$

$S_T = S_L + B_1 + B_2$

$V = \frac{h}{3}\left(B_1 + \sqrt{B_1 B_2} + B_2\right)$

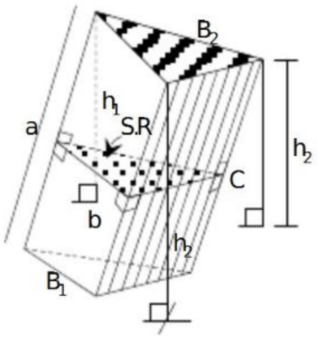


2. Tronco de prisma triangular oblicuo

$S_L = \sum \left(\text{área caras laterales}\right)$

$S_T = S_L + B_1 + B_2$

$V = A_1\left(\frac{a+b+c}{3}\right) \quad V = b_1\left(\frac{h_1+h_2+h_3}{3}\right)$



Tronco de cono

$S_L = \pi\left(R+r\right) g$

$S_T = S_L + \pi R^2 + \pi r^2$

$V = \frac{\pi h}{3}\left[R^2 + Rr + r^2\right]$