INVERSO Multiplicativo

```
#include<bits/stdc++.h>
#define Mod 100000007
#define ll long long int
using namespace std;
char Palavras[10002];
int letras[27];

ll Fat[10002];

void fat()
{
   Fat[0]=1;
   for (ll i = 1; i <= 10002; i++)
   {
         Fat[i] = (Fat[i-1] * i)%Mod;

   }
}
ll mdc(ll  a, ll b, ll *x, ll *y)
{
  ll xx, yy, d;
  if(b==0)
  {
    *x=1; *y=0;
    return a;
  }

  d = mdc(b, a%b, &xx, &yy);
  *x = yy;
  *y = xx - a/b*yy;
```

```
  return d;
}
ll inv5(ll a)
{
  ll x,y,d;
  d = mdc(a,Mod,&x,&y);

  if(x<0)
    x = x+Mod;
  return x;
}
int main()
{
   ll  Numerador;
   fat();

   while(scanf("%s",&Palavras)!=EOF)
   {
      ll Denominador=1;

      memset(letras,0,sizeof(letras));
     if(strcmp(Palavras,"0")==0)
         break;
      for(ll i=0;i<strlen(Palavras);i++)
         letras[Palavras[i]-'a']++;

      Numerador=Fat[strlen(Palavras)];
      for(int i=0;i<26;i++)
      {
         if(letras[i]> 1)
            Denominador = (Denominador * Fat[letras[i] ]) % Mod;
```

```
        }
        Denominador=inv5(Denominador);
        printf("%lld\n",(Numerador*Denominador)%Mod);
    }
}
```

Alpinista

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

using namespace std;
typedef pair<int, int> ii;
typedef vector<int> vi;
// adj.first = pai, adj.second = custo.
ii adj[100010];
// custo entre um dado vertice e o vertice 1.
int cost[100010];
// se um vertice esta no caminho ate o amigo mais distante.
bool path[100010];
// os amigos que voce deseja visitar.
int friends[100010];
// funcao que calcula o custo de ir de um dado vertice ate o vertice 1
// funcao que calcula o custo entre cada vertice u e o vertice 1.
int f(int u)
{
        if(u == 1) return 0;
        else if(cost[u]) return cost[u];
        else return cost[u] = f(adj[u].first)+adj[u].second;
}
int main()
{
        // variaveis de entrada.
```

```
int n, k;
int a, b, c;
adj[1].first = 1;
scanf("%d %d", &n, &k);
        // construindo o grafo.
        for(int i=0; i<n-1; i++)
        {
                scanf("%d %d %d", &a, &b, &c);

                adj[b].first = a;
                adj[b].second = c;
        }

        // lendo os amigos.
        for(int i=0; i<k; i++)
                scanf("%d", &friends[i]);



        // calculando o custo de ir de um dado vertice ao vertice 1
        memset(cost, 0, sizeof(cost));
        for(int i=2; i<=n; i++)
        {
                if(!cost[i])
                  cost[i] = f(adj[i].first)+adj[i].second;
        }



        // encontra o amigo que esta mais longe do vertice 1
        int far = friends[0];
        for(int i=1; i<k; i++)
                far = (cost[ friends[i] ] > cost[far] ? friends[i] : far);
```

```
        // percorre o caminho e o marca.
        memset(path, false, sizeof(path)); path[1] = true;
        do
        {
                path[far] = true;
                far = adj[far].first;
        } while(far != 1);

        // adicionando os custos de ir de todos os vertices que tem um amigo
        // para qualquer vertice que eh parte do caminho.
        // apos visitar o vertice, o adicionamos para o melhor caminho.
        int answer = 0;
        for(int i=0; i<k; i++)
        {
                int aux = friends[i];

                while(!path[aux])
                {
                        answer += adj[aux].second;
                        path[aux] = true;
                        aux = adj[aux].first;
                }
        }

        // imprime a resposta.
        printf("%d\n", answer);
}
```

Backtraking – Permutações

```
#include <stdio.h>


void troca(int vetor[], int i, int j)
{
	int aux = vetor[i];
	vetor[i] = vetor[j];
	vetor[j] = aux;
}


void permuta(int vetor[], int inf, int sup)
{
	if(inf == sup)
	{
		for(int i = 0; i <= sup; i++)
			printf("%d ", vetor[i]);
		printf("\n");
	}
	else
	{
		for(int i = inf; i <= sup; i++)
		{
			troca(vetor, inf, i);
			permuta(vetor, inf + 1, sup);
			troca(vetor, inf, i); // backtracking
		}
	}
}


int main(int argc, char *argv[])
{
```

```
        int v[] = {1, 2, 3, 4};
        int tam_v = sizeof(v) / sizeof(int);

        permuta(v, 0, tam_v - 1);

        return 0;
}
```

Backtraking – Combinaçõe

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

using namespace std;

bool v_bool[21];
int vAtk[21];
int vDef[21];
int vHab[21];
int AtkP;
int DefP;
int HabP;
bool prassodia=false;

void combinacoes(int v[], bool v_bool[], int i, int tam_v)
{
   if(i == tam_v)
   {
      int somaAtk=0;
      int somaDef=0;
      int somaHab=0;
```

```
        int cont=0;
        for(int j = 0; j < tam_v; j++)
        {
           if(v_bool[j])
           {
              cont++;
              somaAtk+=vAtk[j];
              somaDef+=vDef[j];
              somaHab+=vHab[j];
           }

        }

        if(somaAtk==AtkP && somaDef==DefP && somaHab==HabP && cont>1)
        {
           prassodia=true;
           return;
        }


     }
     else
     {
        v_bool[i] = true;
        combinacoes(v, v_bool, i + 1, tam_v);
        v_bool[i] = false;
        combinacoes(v, v_bool, i + 1, tam_v);
     }
  }

  int main()
```

```
{
   int N;



   scanf("%d %d %d %d",&N,&AtkP,&DefP,&HabP);


   for(int i=0;i<N;i++)
      scanf("%d %d %d",&vAtk[i],&vDef[i],&vHab[i]);



   combinacoes(vAtk, v_bool, 0, N);


   if(prassodia)
      printf("Y\n");
   else
      printf("N\n");


   return 0;
}
```

Bases numéricas (32 nesse caso)

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


using namespace std;


int main()
{
   unsigned long long int Num;
   int resto;


   string digitos = "0123456789ABCDEFGHIJKLMNOPQRSTUV";
   string Saida;
```

```
while (cin>>Num)
{
   if(Num==0)
   {
      cout<<"0"<<endl;
      break;
   }
     Saida.clear();
     while(Num >= 32)
                          {

                                  resto = (int)(Num % 32);
                                  Num /= 32;

                                  Saida += digitos[resto];
                          }

                          Saida += digitos[(int)Num];

      for(long int m=Saida.size()-1;m>=0;m--)
      {
         cout<<Saida[m];
      }


  cout<<endl;
}
    return 0;

       }
```

Bellman Ford (menor caminho, com aresta negativa)

```
#include <algorithm>
#include <cstdio>
#include <vector>
#include <queue>
using namespace std;

typedef pair<int, int> ii;
typedef vector<int> vi;
typedef vector<ii> vii;
#define INF 1000000000

int main() {
 int V, E, s, u, v, w;
 vector<vii> AdjList;
 scanf("%d %d %d", &V, &E, &s);

 AdjList.assign(V, vii()); // assign blank vectors of pair<int, int>s to AdjList
 for (int i = 0; i < E; i++) {
  scanf("%d %d %d", &u, &v, &w);
  AdjList[u].push_back(ii(v, w));
 }

 // Bellman Ford routine
 vi dist(V, INF); dist[s] = 0;
 for (int i = 0; i < V - 1; i++)  // relax all E edges V-1 times, overall O(VE)
  for (int u = 0; u < V; u++)                  // these two loops = O(E)
   for (int j = 0; j < (int)AdjList[u].size(); j++) {
    ii v = AdjList[u][j];       // we can record SP spanning here if needed
```

```
      dist[v.first] = min(dist[v.first], dist[u] + v.second);        // relax
    }

 bool hasNegativeCycle = false;
 for (int u = 0; u < V; u++)                    // one more pass to check
  for (int j = 0; j < (int)AdjList[u].size(); j++) {
    ii v = AdjList[u][j];
    if (dist[v.first] > dist[u] + v.second)              // should be false
      hasNegativeCycle = true;     // but if true, then negative cycle exists!
  }
 printf("Negative Cycle Exist? %s\n", hasNegativeCycle ? "Yes" : "No");

 if (!hasNegativeCycle)
  for (int i = 0; i < V; i++)
    printf("SSSP(%d, %d) = %d\n", s, i, dist[i]);

 return 0;
}
```

Floyd Warshall + Bitmask

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

using namespace std;



int main()
{

  int n;

   while(scanf("%d",&n), n)
```

```
{

   vector<vector<int> >grafo(n+1,vector<int>(n+1,0));

   int a,b;
   while(scanf("%d %d",&a,&b),a,b)
   {
       string emp;
       cin>>emp;

       int bitmask = 0;

     for(int i = 0 ; i < (int)emp.size();i++)
     {
       bitmask = bitmask | (1 << emp[i] - 97);
     }
     grafo[a][b] |= bitmask;
   }

   for (int k = 1; k <= n; k++)
   {
       for (int i = 1; i <= n; i++)
       {
                 for (int j = 1; j <= n; j++)
                                          grafo[i][j] = grafo[i][j] | (grafo[i][k] & grafo[k][j]);
       }

   }


   while(scanf("%d %d",&a,&b),a,b)
```

```
      {
          if(!grafo[a][b])
          printf("-\n");
          else
          {
          for(int i = 0;i <=26;i++)
          {
                    if(grafo[a][b] >> i & 1)
                    printf("%c",i+'a');
          }
          printf("\n");
          }
      }
      printf("\n");
   }
}
```

Detecção de Pontes

```
int v, time_s, visit[MAX];
vector<int> ADJ[MAX];
int cont = 0;
int dfs(int u, int pai)
{
   int menor = visit[u] = time_s++;
   int filhos = 0;
   for(int i = 0; i<ADJ[u].size(); i++)
   {
     if(visit[ADJ[u][i]]==0)
     {
       filhos++;
       int m = dfs(ADJ[u][i], u);
```

```
            menor = min(menor,m);
            if(visit[u]<m){
                //ponte
                cont++;
            }
        }
        else if(ADJ[u][i]!=pai)
            menor = min(menor, visit[ADJ[u][i]]);

    }
    return menor;
}


int main()
{

    int a;

    while(scanf("%d %d",&v,&a)!=EOF)
    {
        cont = 0;

        for(int i=0;i<a;i++)
        {
            int a,b;
            scanf("%d %d",&a,&b);
            ADJ[a].push_back(b);
            ADJ[b].push_back(a);

        }
```

```
      time_s = 1;
      memset(visit, 0,sizeof(visit));
      dfs(1,-1);

      for(int i=1;i<=v;i++)
         ADJ[i].clear();

      printf("%d\n",cont);
   }
}
```

Grafo em grade (dijkstra)

```
char grade[502][502];

int Dx [4] = {0, 0, 1,-1};
int Dy [4] = {1,-1, 0, 0};

int n,m;

int dijkstra(pair<int,int>inicio,pair<int,int>fim)
{
    int dist[502][502];

    for(int i = 0 ; i < n ; i++)
    {
     for(int j = 0 ; j < m ; j++)
        dist[i][j] = INF;
    }
    priority_queue < pair<int,pair<int,int> >,
```

```
            vector<pair<int,pair<int,int> > >, greater<pair<int,pair<int,int>  > > > pq;

    pq.push(make_pair(0,make_pair(inicio.first,inicio.second)));
    dist[inicio.first][inicio.second] = 0;

    while(!pq.empty())
    {
       pair<int,pair<int,int> > aux = pq.top();
       pq.pop();

       pair<int,int> vertex = aux.second;

       if(aux.first > dist[vertex.first][vertex.second])
        continue;

        for(int i = 0 ; i < 4 ; i++)
        {
          if (Dx[i] + vertex.first >= 0 && Dx[i] + vertex.first<n && Dy[i]+vertex.second >=0 &&
Dy[i]+vertex.second < m)
          {
             int x = Dx[i] + vertex.first;
             int y = Dy[i] + vertex.second;
             int custo = 0;

             if(grade[x][y] == '#')
                    continue;

             if(isdigit(grade[x][y]))
                  custo = grade[x][y] - '0';

             if(dist[x][y] > (dist[vertex.first][vertex.second] + custo))
```

```
            {
              dist[x][y] = dist[vertex.first][vertex.second] + custo;
              pq.push(make_pair(dist[x][y],make_pair(x,y)));
            }
          }
        }

    }
    return dist[fim.first][fim.second];
}

int main()
{


        scanf("%d%d",&n,&m);

        pair<int,int>inicio;
        pair<int,int>fim;

        for(int i = 0 ; i < n ; i++)
        {
                for(int j = 0 ; j < m ;j++)
                {
                        scanf(" %c",&grade[i][j]);
                        if(grade[i][j] == 'H')
          inicio = make_pair(i,j);
    else if(grade[i][j] == 'E')
          fim = make_pair(i,j);
                }
        }
```

```
            int ans = dijkstra(inicio,fim);

  if(ans == INF)
    printf("ARTSKJID\n");
  else
    printf("%d\n",ans);
}
```

Distancia de Edição

```
int editDist(string str1, string str2, int m, int n)
{




   int **dp = (int **)malloc(MAX * sizeof(int*));

   for(int i=0;i<MAX;i++)
         dp[i] = (int *)malloc(MAX*sizeof(int));

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

         if (i==0)
            dp[i][j] = j;

         else if (j==0)
```

```
            dp[i][j] = i;

        else if (str1[i-1] == str2[j-1])
            dp[i][j] = dp[i-1][j-1];

        else
            dp[i][j] = 1 + min(dp[i][j-1],dp[i-1][j],dp[i-1][j-1]);
      }
    }

    return dp[m][n];
}
```

Fatoração em primos

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

#define lim 1000000
using namespace std;

int main()
{

    long long int X;
    unsigned long long int ans;
    int N;
    scanf("%d",&N);

     int k=1;
    while(N--)
```

```
    {
       ans=1;
        scanf("%lld",&X);
        for(int i=0;i<3401 && X!=1;i++)
        {
           if(X%Primos[i]==0)
           {
              unsigned long long cont=0;
              while(X%Primos[i]==0 && X!=1)
              {
                 cont++;
                 X/=Primos[i];
              }
              if(cont%2!=0)
                cont++;
              ans*=pow((double)Primos[i],(double)cont);
           }
        }

        if(X!=1)
          ans*=(X*X);
          printf("Caso #%d: %llu\n",k++,ans);
        }
   return 0;
}
```

Fibonacci Dinâmico

```
#include<bits/stdc++.h>
#define llu unsigned long long int
```

```
using namespace std;

llu Fibonacci[10002];

llu Fib(llu X)
{
   if(X==0 || X==1)
      return 1;
   if(!Fibonacci[X])
      Fibonacci[X]=Fib(X-1)+Fib(X-2);

   return Fibonacci[X];

}


int main()
{
   llu X;
   while(cin>>X)
      cout<<"Fibonacci["<<X<<"] = "<<Fib(X)<<endl;
}
```

Flood Fill

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

using namespace std;

int dr[] = {1,1,0,-1,-1,-1, 0, 1}; // trick to explore an implicit 2D grid
int dc[] = {0,1,1, 1, 0,-1,-1,-1}; // S,SE,E,NE,N,NW,W,SW neighbors
```

```
int floodfill(int r, int c, char c1, char c2) // returns the size of CC
{
  if (r < 0 || r >= R || c < 0 || c >= C) return 0; // outside grid
  if (grid[r][c] != c1) return 0; // does not have color c1
  int ans = 1; // adds 1 to ans because vertex (r, c) has c1 as its color
  grid[r][c] = c2; // now recolors vertex (r, c) to c2 to avoid cycling!
  for (int d = 0; d < 8; d++)
    ans += floodfill(r + dr[d], c + dc[d], c1, c2);
  return ans; // the code is neat due to dr[] and dc[]
}
```

Floyd Warshall

```
for (int k = 0; k < V; k++) // remember that loop order is k->i->j
for (int i = 0; i < V; i++)
for (int j = 0; j < V; j++)
AdjMat[i][j] = min(AdjMat[i][j], AdjMat[i][k] + AdjMat[k][j]);
```

Checar se grafo é bipartido

```
#include<bits/stdc++.h>
#define INF 1002
using namespace std;



int cont = 0;
bool nao = false;
```

```
int main()
{
   int N;
   char aux[5];
   while(scanf("%d",&N),N)
   {
      getchar();
      list<int>adj[102];


      for(int i=0;i<N;i++)
      {
         int a;
         scanf("%d",&a);
         getchar();
         char X[1000];
         char *C;
         scanf("%1000[^\n]",X);
         getchar();

         C=strtok(X," ");
         while(C!=NULL)
         {

           int  b = strtol(C,0,10);


            adj[a-1].push_back(b-1);
            adj[b-1].push_back(a-1);

            C = strtok (NULL, " ");
```

```
    }

}
queue<int>q;
q.push(0);

vector<int>colors(N,INF);
colors[0] = 0;
bool EhBipartido =true;

list<int>::iterator it;

while(!q.empty() && EhBipartido)
{
   int u = q.front();
   q.pop();
   for(it = adj[u].begin();it!=adj[u].end();it++)
   {
      int v = *it;
      if(colors[v] == INF)
      {
         colors[v] = 1 - colors[u];
         q.push(v);
      }
      else if(colors[u] == colors[v])
      {
         EhBipartido = false;
         break;
      }
   }
```

```
        }

        if(EhBipartido)
            printf("SIM\n");
        else
            printf("NAO\n");
    }
}
```

Envoltória Convexa

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

using namespace std;

vector<pair<int,int> >cords;

struct point
{
    int x, y;
    inline int sqr(int k)
    {
        return (k*k);
    }
    int d2(point p)
    {
        return sqr(x - p.x) + sqr(y - p.y);
    }
    point(int _x=0, int _y=0):x(_x),y(_y){}
    point operator-(point p)
    {
        return point(x-p.x,y-p.y);
    }
```

```
    int operator*(point p)
    {
        return (x*p.y)-(y*p.x);
    }
}p[2048];
int N;
bool ord(point a, point b)
{
    int k = (a - p[0]) * (b - p[0]);
    return k ? k > 0 : p[0].d2(a) < p[0].d2(b);
}

bool grahamScan()
{
    int idx, miny = 1<<25, t = 0, x;
    point *q[2048];
    for(int i = 0; i < N; ++i)
    {
        if(i == N) break;
        if(p[i].y < miny || (p[i].y == miny && p[i].x < x))
            miny = p[i].y, x = p[i].x, idx = i;
    }
    if(N < 3) return 0;
    swap(p[0],p[idx]);
    sort(p+1, p+N, ord);
    q[t++] = &p[0];
    q[t++] = &p[1];
    for(int i = 2; i < N; ++i)
    {
        while(t > 1 && ((*q[t-1]- *q[t-2])*(p[i] - *q[t-2])) <= 0) t--;
        q[t++] = &p[i];
```

```
    }

    for(int x = 0; x < t; ++x)
     cords.push_back(make_pair(q[x]->x,q[x]->y));




}
double distancia(int x0,int y0,int x1,int y1)
{
    return sqrt(((x1 - x0)*(x1 - x0)) + ((y1 - y0)*(y1 - y0)));
}
int main(){
    int a, b, c;



    while(scanf("%d", &a),a)
    {
        cords.clear();
        double Total = 0;

        for(N = 0; N < a; ++N)
            scanf("%d %d", &p[N].x, &p[N].y);

        grahamScan();

        for(int i=0;i<cords.size();i++)
        {
            int x0 = cords[i].first;
            int y0 = cords[i].second;
            int x1,y1;
```

```
        if(i<cords.size()-1)
        {
           x1 = cords[i+1].first;
           y1 = cords[i+1].second;
        }
        else
        {
           x1 = cords[0].first;
           y1 = cords[0].second;
        }


        Total += distancia(x0,y0,x1,y1);

      }
      printf("Tera que comprar uma fita de tamanho ");
      cout<<fixed<<setprecision(2);
      cout<<fixed<<Total<<"."<<endl;
   }
   return 0;
}
```

Josephus Problem

```
/// n -> Número de pessoas na roda
/// k -> fator de morte

///Se o fator de mortes (k) for igual a 2:

int Josephus(int n)
{
```

```
   int r = 2*(n-pow(2,(int)log2(n))) + 1;

   return r;

}

///Se não:

///Versão iterativa,para evitar stack overflows:

int Josephus(int n, int k)
{
   int ans = 0;
   for (int i = 2; i <= n; ++i)
      ans = (ans + k) % i;

   return ans;
}

/// Versão recursiva:

int Josephus(int n, int k)
{

   if (n == 1)
      return 0;
   return (Josephus(n - 1, k) + k) % n;
}
```

LCA (Menor ancestral comum)

```
#include <stdio.h>
#include <string.h>
#include <vector>
#include <math.h>
using namespace std;
const int MAXN = 50010;

int n;
int carta[MAXN];
vector<int> pos[MAXN/2];

vector<int> adj[MAXN];

int pai[MAXN];
int nivel[MAXN];
int super_pai[MAXN];

int maior_altura;
void monta_arvore(int u, int p, int l);

int segmento;
void monta_super_pai(int u, int p);

int lca_2(int a, int b);

int main() {
	scanf("%d", &n);
	for(int i=1; i<=n; i++) {
		scanf("%d", &carta[i]);
```

```
            pos[ carta[i] ].push_back(i);
        }
        for(int i=1; i<n; i++) {
            int a, b;
            scanf("%d %d", &a, &b);

            adj[a].push_back(b);
            adj[b].push_back(a);
        }

        maior_altura = 0;
        monta_arvore(1, 1, 0);
        segmento = sqrt(maior_altura);

        monta_super_pai(1, 1);

        int res = 0;
        for(int i=1; i<=n/2; i++) {
            int a = pos[i][0];
            int b = pos[i][1];

            int c = lca_2(a, b);

            res += (nivel[a]-nivel[c]) + (nivel[b]-nivel[c]);
        }

        printf("%d\n", res);
}

void monta_arvore(int u, int p, int l) {
        pai[u] = p;
```

```
	nivel[u] = l;

	if(l > maior_altura) {
		maior_altura = l;
	}

	for(int i=0; i<(int)adj[u].size(); i++) {
		int v = adj[u][i];

		if(!pai[v]) {
			monta_arvore(v, u, l+1);
		}
	}
}

void monta_super_pai(int u, int p) {
	super_pai[u] = p;

	if(nivel[u]%segmento == 0) {
		p = u;
	}

	for(int i=0; i<(int)adj[u].size(); i++) {
		int v = adj[u][i];

		if(!super_pai[v]) {
			monta_super_pai(v, p);
		}
	}
}
```

```
int lca_2(int a, int b) {
	while(super_pai[a] != super_pai[b]) {
		if(nivel[a] > nivel[b]) {
			a = super_pai[a];
		} else {
			b = super_pai[b];
		}
	}
	while(a != b) {
		if(nivel[a] > nivel[b]) {
			a = pai[a];
		} else {
			b = pai[b];
		}
	}
	return a;
}
```

LCS (Maior subsequência comum)

```
/*
	LCS - longest common subsequence
	What's the length of the LCS?
	Example: ABCB and BCAB
	LCS is BCB (not necessarily contiguous)
	Answer: 3
*/

#include <iostream>
#include <algorithm>
using namespace std;
```

```
int get_len_lcs(string& s1, string& s2)
{
	int len_s1 = s1.size(), len_s2 = s2.size();

	// matrix
	int mat[len_s1 + 1][len_s2 + 1];

	// initializes the first line and column with 0
	for(int i = 1; i <= len_s1; i++)
		mat[i][0] = 0;
	for(int i = 0; i <= len_s2; i++)
		mat[0][i] = 0;

	for(int i = 1; i <= len_s1; i++)
	{
		for(int j = 1; j <= len_s2; j++)
		{
			if(s1[i - 1] == s2[j - 1])
				mat[i][j] = mat[i - 1][j - 1] + 1;
			else
				mat[i][j] = max(mat[i][j - 1], mat[i - 1][j]);
		}
	}

	cout << "\nMatrix:\n\n";
	for(int i = 0; i <= len_s1; i++)
	{
		for(int j = 0; j <= len_s2; j++)
			cout << mat[i][j] << " ";
		cout << "\n";
	}
```

```
	return mat[len_s1][len_s2];
}

int main(int argc, char *argv[])
{
	string s1("AGGTAB"), s2("GXTXAYB");

	int len_lcs = get_len_lcs(s1, s2);

	cout << "\nLength: " << len_lcs << endl;

	return 0;
}
```

Maior palíndromo subsequente

```
// Longest Palindromic Subsequence - Dynamic Programming

#include <iostream>
#include <algorithm>

using namespace std;

int lps(string & s, int size_s)
{
	int mat[size_s][size_s];

	// strings de tamanho 1 são palíndromos de tamanho 1
	for(int i = 0; i < size_s; i++)
		mat[i][i] = 1;
```

```
	int tam = 2;

	while(tam <= size_s)
	{
		for(int i = 0, j = tam - 1 + i; j < size_s; i++, j++)
		{
			if(s[i] != s[j])
				mat[i][j] = max(mat[i][j - 1], mat[i + 1][j]);
			else
				mat[i][j] = mat[i + 1][j - 1] + 2;
		}
		tam++;
	}

	return mat[0][size_s - 1];
}

int main(int argc, char *argv[])
{
	string s("ATCATA");

	cout << lps(s, s.size()) << endl;

	return 0;
}
```

Maior Subsequencia aumentando (LIS)

```
/* Dynamic Programming C/C++ implementation of LIS problem */
#include<stdio.h>
#include<stdlib.h>
```

```
/* lis() returns the length of the longest increasing
   subsequence in arr[] of size n */
int lis( int arr[], int n )
{
   int *lis, i, j, max = 0;
   lis = (int*) malloc ( sizeof( int ) * n );

   /* Initialize LIS values for all indexes */
   for ( i = 0; i < n; i++ )
      lis[i] = 1;

   /* Compute optimized LIS values in bottom up manner */
   for ( i = 1; i < n; i++ )
      for ( j = 0; j < i; j++ )
         if ( arr[i] > arr[j] && lis[i] < lis[j] + 1)
            lis[i] = lis[j] + 1;

   /* Pick maximum of all LIS values */
   for ( i = 0; i < n; i++ )
      if ( max < lis[i] )
         max = lis[i];

   /* Free memory to avoid memory leak */
   free( lis );

   return max;
}

/* Driver program to test above function */
int main()
{
```

```
  int arr[] = { 10, 22, 9, 33, 21, 50, 41, 60 };
  int n = sizeof(arr)/sizeof(arr[0]);
  printf("Length of LIS is %d\n", lis( arr, n ) );
  return 0;
}
```

Maior pilha de caixas (LIS)

```
/* Dynamic Programming implementation of Box Stacking problem */
#include<stdio.h>
#include<stdlib.h>

/* Representation of a box */
struct Box
{
  // h –> height, w –> width, d –> depth
  int h, w, d;  // for simplicity of solution, always keep w <= d
};

// A utility function to get minimum of two intgers
int min (int x, int y)
{ return (x < y)? x : y; }

// A utility function to get maximum of two intgers
int max (int x, int y)
{ return (x > y)? x : y; }

/* Following function is needed for library function qsort(). We
   use qsort() to sort boxes in decreasing order of base area.
   Refer following link for help of qsort() and compare()
   http://www.cplusplus.com/reference/clibrary/cstdlib/qsort/ */
int compare (const void *a, const void * b)
{
```

```
   return ( (*(Box *)b).d * (*(Box *)b).w ) - ( (*(Box *)a).d * (*(Box *)a).w );
}

/* Returns the height of the tallest stack that can be formed with give type of boxes */
int maxStackHeight( Box arr[], int n )
{
  /* Create an array of all rotations of given boxes
     For example, for a box {1, 2, 3}, we consider three
     instances{{1, 2, 3}, {2, 1, 3}, {3, 1, 2}} */
  Box rot[3*n];
  int index = 0;
  for (int i = 0; i < n; i++)
  {
    // Copy the original box
    rot[index] = arr[i];
    index++;

    // First rotation of box
    rot[index].h = arr[i].w;
    rot[index].d = max(arr[i].h, arr[i].d);
    rot[index].w = min(arr[i].h, arr[i].d);
    index++;

    // Second rotation of box
    rot[index].h = arr[i].d;
    rot[index].d = max(arr[i].h, arr[i].w);
    rot[index].w = min(arr[i].h, arr[i].w);
    index++;
  }

  // Now the number of boxes is 3n
```

```
n = 3*n;

/* Sort the array 'rot[]' in decreasing order, using library
   function for quick sort */
qsort (rot, n, sizeof(rot[0]), compare);

// Uncomment following two lines to print all rotations
// for (int i = 0; i < n; i++ )
//    printf("%d x %d x %d\n", rot[i].h, rot[i].w, rot[i].d);

/* Initialize msh values for all indexes
   msh[i] –> Maximum possible Stack Height with box i on top */
int msh[n];
for (int i = 0; i < n; i++ )
   msh[i] = rot[i].h;

/* Compute optimized msh values in bottom up manner */
for (int i = 1; i < n; i++ )
   for (int j = 0; j < i; j++ )
      if ( rot[i].w < rot[j].w &&
           rot[i].d < rot[j].d &&
           msh[i] < msh[j] + rot[i].h
         )
      {
           msh[i] = msh[j] + rot[i].h;
      }


/* Pick maximum of all msh values */
int max = -1;
for ( int i = 0; i < n; i++ )
```

```
    if ( max < msh[i] )
        max = msh[i];

  return max;
}

/* Driver program to test above function */
int main()
{
  Box arr[] = { {3, 3, 3}, {2, 2, 2}};
  int n = sizeof(arr)/sizeof(arr[0]);

  printf("The maximum possible height of stack is %d\n",
        maxStackHeight (arr, n) );

  return 0;
}
```

Maior pilha de cilindros

```
//Ajude o seu barriga
#include<bits/stdc++.h>



using namespace std;

struct cilindros
{
   int raio;
   int altura;
   int cor;
};
```

```
int n;
int doit(cilindros p[])
{

  int pilha[n];

  for(int i = 0 ; i < n ; i++)
  {
     pilha[i] = p[i].altura;

  }
  for(int i = 0 ; i < n ; i++)
  {
     for(int j = 0 ; j < i; j++)
     {
        if((p[i].raio < p[j].raio) && (pilha[i] < (pilha[j] + p[i].altura)))
        {
           if(p[i].cor == 1 && p[j].cor != 2)
              pilha[i] = pilha[j] + p[i].altura;
           else if(p[i].cor == 2 && p[j].cor != 4)
              pilha[i] = pilha[j] + p[i].altura;
           else if(p[i].cor == 4 && p[j].cor != 3)
              pilha[i] = pilha[j] + p[i].altura;
           else if(p[i].cor == 3 && p[j].cor != 1)
              pilha[i] = pilha[j] + p[i].altura;

        }
     }
  }
  int max = -1;
  for ( int i = 0; i < n; i++ )
```

```
    if ( max < pilha[i] )
       max = pilha[i];

 return max;

}

bool comp(cilindros i, cilindros j)
{
        if(i.raio > j.raio)
                return true;
        else
                return false;
}

int main()
{


   map<string,int>cores;
   cores["VERMELHO"] = 1;
   cores["LARANJA"] = 2;
   cores["VERDE"] = 3;
   cores["AZUL"] = 4;

   while(scanf("%d",&n),n)
   {
        cilindros p[n];
        for(int i = 0 ; i < n ; i++)
        {
       int h,r;
```

```
        string c;


        cin>>h>>r>>c;
        p[i].cor = cores[c];
        p[i].altura = h;
        p[i].raio = r;
          }
          sort(p,p+n,comp);


      int ans = doit(p);


      printf("%d centimetro(s)\n",ans);
   }
}
```

Mochila (saber o que foi usado nela)

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

using namespace std;

int a[10002];
int b[10002];




void mochila(int a[], int b[], int W, int n)
{
   int i,w;
   int K[n+1][W+1];
   for (i = 0; i <= n; i++)
  {
     for (w = 0; w <= W; w++)
```

```
      {
         if (i==0 || w==0)
            K[i][w] = 0;
         else if (b[i-1] <= w)
              K[i][w] = max(a[i-1] + K[i-1][w-b[i-1]],  K[i-1][w]);
         else
              K[i][w] = K[i-1][w];
      }
   }
   int PesoTotal = W;
   int TotalBrinquedos  = K[n][W];
   int Usados = 0;
   for(int i=n;i>0;i--)
   {
      if(K[i-1][W] != K[i][W])
      {
         W -= b[i-1];
         Usados++;
      }
   }
   printf("%d brinquedos\n",TotalBrinquedos);
   printf("Peso: %d kg\n",PesoTotal-W);
   printf("sobra(m) %d pacote(s)\n",n-Usados);
}

int main()
{
    int T;
    scanf("%d",&T);

    while(T--)
```

```
  {
    int pac;
    scanf("%d",&pac);
    for(int i=0;i<pac;i++)
      scanf("%d %d",&a[i],&b[i]);

    mochila(a,b,50,pac);
    printf("\n");
  }
}
```

Algoritmo da Moeda

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

using namespace std;

int pval[1000002];
int val, nmoedas;
int vmoedas[1000002];

int main(){

  int n;
  scanf("%d",&n);

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

        cin >> nmoedas;
        cin >> val;
        memset(pval,0,sizeof(pval));
```

```
                for (int i = 0; i < nmoedas; i++)
    {
                        cin >> vmoedas[i];
                        pval[vmoedas[i]] = 1;
                }

                for (int i = 1; i <= val; i++)
    {
                        if (pval[i])
      for (int a = 0; a < nmoedas; a++)
      {
                                        if (i + vmoedas[a] > val)
                                          break;
                                        if (pval[i + vmoedas[a]] == 0)
            pval[i + vmoedas[a]] = pval[i] + 1;
                                        else
                                          pval[i + vmoedas[a]] = min(pval[i +
vmoedas[a]],pval[i] + 1);
                        }
                }

                if (pval[val]) cout << pval[val] << endl;




       }

       return 0;
}
```

Iterações de uma DFS

```
int dfs(int vertex, int v)
{
   int i, count = 0;

   discovered[vertex] = true;

   for (i = 0; i < v; i++)
   {
      if (graph[vertex][i] && !discovered[i])
         count += dfs(i, v) + 1;
   }
   return count;
}
```

Algoritmo de Prim (MST)

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

using namespace std;

typedef pair<int, int> ii;
typedef vector<int> vi;
typedef vector<ii> vii;
typedef vector<vector<ii> > Graph;
int n;

vi taken;
```

```
priority_queue<ii> pq;

void process(int vtx,Graph &AdjList)
{
  taken[vtx] = 1;
  for (int j = 0; j < (int)AdjList[vtx].size(); j++)
  {
    ii v = AdjList[vtx][j];
    if (!taken[v.first])
        pq.push(ii(-v.second, -v.first));
  }
}


int main()
{

    long long int mst_cost = 0;

    scanf("%d",&n);
    Graph AdjList(n+2);
     for(int i = 1;i<n;i++)
     {
        int x;
        scanf("%d",&x);

        for(int j = 0 ; j < x;j++)
        {
           int y,p;
           scanf("%d %d",&y,&p);
           AdjList[i].push_back(make_pair(y,p));
```

```
            AdjList[y].push_back(make_pair(i,p));
        }
      }
      taken.assign(n+1, 0);
      int cont = 0;
     for(int i =  1 ; i <= n;i++ )
     {

        if(!taken[i])
        {
            cont++;
            process(i,AdjList);
            while (!pq.empty())
            {
                ii front = pq.top(); pq.pop();
                int u = -front.second;
                int w = -front.first;
                if (!taken[u])
                {
                    mst_cost += w;
                    process(u,AdjList);
                }
            }
        }
     }

   printf("%d %lld\n",cont, mst_cost);

}
```

Recuperando uma árvore binária

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



using namespace std;

/// nesse caso tendo os percurso pre-fixo e in-fixo


void geraPos(string x, string y)

{


   char raiz = x[0];

   int pos = y.find(raiz);


   string p1,p2;

   string e1,e2;


   if(pos)

   {

      p1 = x.substr(1,pos);

      e1 = y.substr(0,pos);

      geraPos(p1,e1);

   }

   if(pos + 1 < x.size())

   {

      p2 = x.substr(pos+1);

      e2 = y.substr(pos+1);

      geraPos(p2,e2);

   }

   printf("%c",raiz);
```

```
}

int main()
{

    string x,y;

    while(cin>>x>>y)
    {
       geraPos(x,y);
       printf("\n");
    }
}
```

STRTOK

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

using namespace std;



bool Comp(string i,string j)
{

   return i.size()>j.size();

}



int main()
```

```
{
  char X[1000];
  vector<string>Y;

  int N;
  char *C;
  cin>>N;
  getchar();
  while(N--)
  {
     scanf("%1000[^\n]",X);
     getchar();
     C=strtok(X," ");
     while(C!=NULL)
     {

        Y.push_back(C);
        C = strtok (NULL, " ");
     }

     stable_sort(Y.begin(),Y.end(),Comp);
     for(int i=0;i<Y.size();i++)
     {
        if(i==Y.size()-1)
        cout<<Y[i]<<endl;
        else
           cout<<Y[i]<<" ";
     }
     Y.clear();
  }
}
```

SubsetSum

```
// A Dynamic Programming solution for subset sum problem
#include <stdio.h>

// Returns true if there is a subset of set[] with sun equal to given sum
bool isSubsetSum(int set[], int n, int sum)
{
   // The value of subset[i][j] will be true if there is a subset of set[0..j-1]
   //  with sum equal to i
   bool subset[sum+1][n+1];

   // If sum is 0, then answer is true
   for (int i = 0; i <= n; i++)
     subset[0][i] = true;

   // If sum is not 0 and set is empty, then answer is false
   for (int i = 1; i <= sum; i++)
     subset[i][0] = false;

    // Fill the subset table in botton up manner
    for (int i = 1; i <= sum; i++)
    {
      for (int j = 1; j <= n; j++)
      {
        subset[i][j] = subset[i][j-1];
        if (i >= set[j-1])
          subset[i][j] = subset[i][j] || subset[i - set[j-1]][j-1];
      }
    }
```

```
   /* // uncomment this code to print table
    for (int i = 0; i <= sum; i++)
    {
      for (int j = 0; j <= n; j++)
         printf ("%4d", subset[i][j]);
      printf("\n");
    } */

    return subset[sum][n];
}

// Driver program to test above function
int main()
{
  int set[] = {3, 34, 4, 12, 5, 2};
  int sum = 9;
  int n = sizeof(set)/sizeof(set[0]);
  if (isSubsetSum(set, n, sum) == true)
     printf("Found a subset with given sum");
  else
     printf("No subset with given sum");
  return 0;
}
```

Tarjan – Detectar todos os ciclos de um grafo

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

using namespace std;

typedef vector<int> vi;
typedef pair<int,int> ii;
```

```
vi dfs_num(100003,0);
vi dfs_low(100003,0);
vi S;
vi visited(100003,0);
int V;
int dfsNumberCounter;

vector<int>AdjList[100003];
int cont = 0;
int contTotal = 0; // conta TODOS ciclos do grafo



void tarjan(int u)
{

        dfs_low[u] = dfs_num[u] = dfsNumberCounter++;
        S.push_back(u);
        visited[u] = 1;
        for (int j = 0; j < (int)AdjList[u].size(); j++)
        {
                int  v = AdjList[u][j];
                if (!dfs_num[v])
                   tarjan(v);
                if (visited[v])
                        dfs_low[u] = min(dfs_low[u], dfs_low[v]);
    }
        if (dfs_low[u] == dfs_num[u])
        {
                cont = 0;
                while (1)
```

```
                        {
                                cont++;
                                int v = S.back(); S.pop_back(); visited[v] = 0;
                                // v = um dos vértices desse ciclo fortemente ligado

                                if (u == v) break;
                        }
                        if(cont > 1)
                                contTotal++;

                }
}
map<string,int>mp;
int get(string x)
{
        if(!mp.count(x))
                mp[x] = mp.size()-1;
        return mp[x];
}

int main()
{


  string x,y;

  while(cin>>x>>y)
  {

    int a = get(x);
    int b = get(y);
```

```
    AdjList[a].push_back(b);
  }


  dfsNumberCounter = 0;

        for (int i = 0; i < (int)mp.size(); i++)
        {
                if (!dfs_num[i])
                  tarjan(i);

        }
        printf("%d\n",contTotal);

}
```

BFS para transformar número em outro (qnt passos)

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

using namespace std;

void bfs(int a,int b)
{
  queue<pair<int,int> > fila;
  map<int,bool> visitado;
  fila.push(pair<int,int>(a,0));
  pair<int,int> par;
```

```
   while(!fila.empty()){
      par = fila.front();
      fila.pop();

      if(visitado[par.first])
         continue;
      visitado[par.first] = true;
      if(par.first == b )
      {
         printf("%d\n",par.second);
         return;
      }
      fila.push(pair<int,int>(par.first*2,par.second+1));
      fila.push(pair<int,int>(par.first*3,par.second+1));
      fila.push(pair<int,int>(par.first/2,par.second+1));
      fila.push(pair<int,int>(par.first/3,par.second+1));
      fila.push(pair<int,int>(par.first+7,par.second+1));
      fila.push(pair<int,int>(par.first-7,par.second+1));
   }

}
int main()
{
   int a,b;
   scanf("%d %d",&a,&b);
   bfs(a,b);
return 0;
}
```

Funçõeszinhas marotas

istringstream (string x) >> inteiro y

transforma uma string em inteiro (strings C++)

sprintf(N,"%d",X)

transforma inteiro(X) em string(N)

X = strtol(N,0,10)

Transforma string(N) em inteiro long (X) (Strings C)

Busca Binária com limite de iterações.

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

using namespace std;



double A;

int cont =0;
void gera(double Vet[],double inic,double maior,double n)
{

  double corte = (maior+inic)/2.0;
  double total = 0;
  cont++;



   for(int i=0;i<n;i++)
   {
      if(Vet[i] > corte)
      {
         total +=  (Vet[i]-corte);
      }


   }
   if(cont == 100 || total == A)
   {
      printf("%.4lf\n",corte);
      return;
```

```
    }


    if(total>A)
        gera(Vet,inic,corte,n);
    else
        gera(Vet,corte,maior,n);

}


int main()
{
    double N;

    while(scanf("%lf %lf",&N,&A),N,A)
    {
        int n = N;
        double Tiras[n];
        double Total = 0;
        double maior;

        for(int i=0;i<N;i++)
        {
            scanf("%lf",&Tiras[i]);
            Total += Tiras[i];
            if(i==0 || Tiras[i]>maior)
                maior = Tiras[i];

        }
        if(Total == A)
```

```
        {
            printf(":D\n");
            continue;
        }
        else if(Total<A)
        {
            printf("-.-\n");
            continue;
        }
        cont = 0;
        gera(Tiras,maior,0,N);
    }
}
```