Editorial Concursos de Programación

Solución a los problemas de los concursos realizados por el grupo de Algoritmia durante el ciclo 105 A.C. - 2015 D.C.

Grupo Universitario de Algoritmia y Programación Competitiva

Universidad Nacional Autónoma de México

Índice general

Introducción

Este es un compendio de editoriales de concursos realizadas por el grupo de Algortimia de la FES Acatlán.

Capítulo 1

3.er Concurso Interno de Programación

1.1. Problema A - Abejas

Dificultad: ⋆⋆
Temas: Geometría, Ordenamiento, Algoritmos Voraces
Complejidad: $O(n \log n)$

Entendiendo el problema

Dado un punto base y n puntos adicionales, quieres saber cuál es el área mínima del círculo que cubre al menos k puntos adicionales con centro en el punto base.

Solución

Siguiendo una estrategia *greedy*, o voraz, podemos ver que el círculo que buscamos es aquél que tiene como radio la distancia al k-ésimo punto más cercano al centro.

Código

La dificultad recae en implementar un algoritmo de ordenamiento cuya complejidad sea $O(n \log n)$ y ordenar por distancias, pero aprovechando las bondades de la `STL` de `C++` tenemos lo siguiente.

```
1 #include <iostream>
2 #include <cstdio>
3 #include <algorithm>
4 #define For(i, a, b) for(int i=a; i<b; ++i)
5 #define PI 3.14159265359
```

```cpp
using namespace std;

struct punto
{
        double x, y, dist;
};

bool myfunction(punto A, punto B){ return A.dist < B.dist; }

int main()
{
        int T;
        cin>>T;
        while (T--)
        {
                double Cx, Cy;
                int N, K;
                string basura;
                punto puntos[200];
                cin>>Cx>>Cy>>N>>K;
                For(i, 0, N)
                {
                        cin>>puntos[i].x>>puntos[i].y;
                        puntos[i].dist = (Cy-puntos[i].y)*(Cy-puntos[i].y) +
 ↪  (Cx-puntos[i].x)*(Cx-puntos[i].x);
                }
                cin>>basura;
                sort(puntos, puntos+N, myfunction);
                printf("%.2lf\n",PI*puntos[K-1].dist);
        }
        return 0;
}
```

1.2. Problema B - Buscando Oro

Dificultad: ⋆⋆⋆⋆
Temas: Grafos, Programación dinámica
Complejidad: $O(k^2 2^k)$

Entendiendo el problema

Dada una cuadrícula de $n \times m$ ($1 \leq n, m \leq 20$), donde existen k ($1 \leq k \leq 15$) posiciones especiales, y una posición inicial, se pide encontrar el mínimo número de pasos necesarios para pasar por todas las k posiciones especiales y regresar a la posición inicial.

Solución

Para resolver el problema hay que notar que lo que necesitamos es un ciclo hamiltoniano mínimo. El conjunto de vértices del grafo serán las k posiciones especiales más la posición inicial, y dos vértices u, v, estarán unidos por una

arista con peso igual al número mínimo de pasos para llegar de u a v. Debido a que para cada posición de la cuadrícula es posible moverse a los 8 vecinos adyacentes, la distancia mínima entre dos posiciónes es la norma uniforme entre ellas. Una vez construido el grafo hay que encontrar el ciclo hamiltoniano mínimo utilizando programación dinámica.

Código

```
#include <iostream>
#include <cstring>
#include <cstdlib>
#include <cstdio>
#include <vector>
#include <map>
#define For(i, a, b) for(int i=a; i<b; ++i)
using namespace std;

struct punto
{
    int x, y;
};

struct nodo
{
    string S;
    int pos[20];
};

int tabla[30][35000];
nodo I[35000];
int dist[30][30], N = 1;
map <string, int> Ind;

int max(int a, int b)
{
    return a > b ? a : b;
}

void getSubStrings(int t, int x, int n, int* pos, string A)
{
    A = A + (char)(x+'0');
    if (t == n)
    {
        Ind[A] = *pos;
        I[*pos].S = A;
        For(k, 1, A.length())
        {
            string B = A;
            int u = int(B[k]-'0');
            B.replace(k, 1, "");
            I[*pos].pos[u] = Ind[B];
        }
        (*pos)++;
        return;
    }

```

```
49      for( ; x < N-1; x++)
50          getSubStrings(t+1, x+1, n, pos, A);
51  }
52
53  void iniTabla()
54  {
55      int pos = 0;
56      For(i, 0, N)
57          getSubStrings(0, 0, i, &pos, "");
58  }
59
60  void HamCycle()
61  {
62      For(i, 0, N)
63          tabla[i][0] = dist[i][0];
64
65      For(j, 1, 1<<(N-1))
66      {
67          For(i, 0, N)
68          {
69              if (j == (1<<(N-1))-1 and i) return;
70              int min = -1;
71              For(k, 1, I[j].S.length())
72              {
73                  int u = int(I[j].S[k]-'0'), S_u = I[j].pos[u];
74                  if (tabla[u][S_u] + dist[u][i] < min or min == -1)
75                      min = tabla[u][S_u] + dist[u][i];
76              }
77              tabla[i][j] = min;
78          }
79      }
80  }
81
82  int main()
83  {
84      //freopen("entrada.in","r",stdin);
85      punto pos[20];
86      string mundo[20];
87      int T;
88      cin>>T;
89      For(casos, 0, T)
90      {
91          int R, C;
92          cin>>R>>C;
93          For(i, 0, R)
94              cin>>mundo[i];
95          For(i, 0, R)
96              For(j, 0, C)
97                  if (mundo[i][j] == 'x')
98                  {
99                      pos[0].x = j;
100                     pos[0].y = i;
101                 }
102                 else if (mundo[i][j] == 'g')
103                 {
104                     pos[N].x = j;
105                     pos[N].y = i;
```

```
106                     N++;
107                 }
108         For(i, 0, N)
109             For(j, i+1, N)
110                 dist[i][j] = dist[j][i] = max(abs(pos[i].x - pos[j].x),
    ↪  abs(pos[i].y - pos[j].y));
111
112         iniTabla();
113         HamCycle();
114         cout<<"Case "<<casos+1<<": "<<tabla[0][(1<<(N-1))-1]<<endl;
115         Ind.clear();
116         N = 1;
117     }
118     return 0;
119 }
```

1.3. Problema C - Cortando Pizza

Dificultad: ⋆⋆
Temas: Matemáticas
Complejidad: $O(1)$

Entendiendo el problema

El problema se puede ver como el máximo número de regiones en las que el plano puede ser cortado con n líneas, el cual es un problema muy conocido.

Solución

Definamos nuestro resultado como $f(n)$. Claramente $f(1) = 2$, y dado que cada nueva línea nos crea n nuevas secciones, ya que divide cada sección en dos, entonces $f(n) = f(n-1) + n$. Por lo tanto:

$$\begin{aligned} f(n) &= n + f(n-1) \\ &= n + n - 1 + f(n-2) \\ &= n + n - 1 + ... + 2 + f(1) \\ &= n + n - 1 + ... + 2 + 1 + 1 \\ &= \sum_{i=1}^{n} i + 1 \\ &= \frac{n(n+1)}{2} + 1 \end{aligned}$$

Código

```
1 #include<iostream>
2 using namespace std;
3 int main()
4 {
```

```
5       long long quesos = 0, n;
6       cin >> n;
7       while(n >= 0)
8       {
9           quesos = n*(n+1)/2 + 1;
10          cout << quesos << endl;
11          cin >> n;
12      }
13      return 0;
14  }
```

1.4. Problema D - Dark Souls

Dificultad: ⋆
Temas: Implementación
Complejidad: $O(nm)$

Entendiendo el problema

En este problema te encuentras en una cuadrícula de $n \times m$ $(2 \leq n, m \leq 30)$, y se te pide saber si necesitas escapar, o no, después de que ciertas casillas de la cuadrícula sean atacadas por los dragones de la cuadrícula. Siempre que existe al menos una casilla en la cuadrícula que no sea atacada por ningún dragón, no es necesario escapar.

Solución

Es posible marcar todas las casillas que ataca cada dragón. Al terminar con todos los dragones se debe recorrer toda la cuadrícula y revisar si existe al menos una casilla no atacada.

Código

```
1   #include <iostream>
2   #include <cstring>
3   #define For(i, a, b) for(int i=a; i<b; ++i)
4   #define FOR(i, a, b) for(int i=a; i>b; --i)
5   using namespace std;
6
7   bool mundo[35][35];
8   int N, M;
9
10  void fuego(int x, int y, char d)
11  {
12      if (d == 'U')
13          FOR(i, y, -1)
14          {
15              mundo[i][x] = 1;
16              if (x-1 >= 0) mundo[i][x-1] = 1;
17              if (x+1 < M) mundo[i][x+1] = 1;
18          }
19      else if (d == 'D')
```

```
20          For(i, y, N)
21          {
22              mundo[i][x] = 1;
23              if (x-1 >= 0) mundo[i][x-1] = 1;
24              if (x+1 < M) mundo[i][x+1] = 1;
25          }
26      else if (d == 'L')
27          FOR(i, x, -1)
28          {
29              mundo[y][i] = 1;
30              if (y-1 >= 0) mundo[y-1][i] = 1;
31              if (y+1 < N) mundo[y+1][i] = 1;
32          }
33      else
34          For(i, x, M)
35          {
36              mundo[y][i] = 1;
37              if (y-1 >= 0) mundo[y-1][i] = 1;
38              if (y+1 < N) mundo[y+1][i] = 1;
39          }
40  }
41
42  void electrico(int x, int y, char d)
43  {
44      mundo[y][x] = 1;
45      if (d == 'U')
46      {
47          FOR(i, y-1, -1) mundo[i][x] = 1;
48          for(int i=y-1, j=x-1; i >= 0 and j >= 0; i--, j--) mundo[i][j] = 1;
49          for(int i=y-1, j=x+1; i >= 0 and j < M; i--, j++) mundo[i][j] = 1;
50      }
51      if (d == 'D')
52      {
53          For(i, y+1, N) mundo[i][x] = 1;
54          for(int i=y+1, j=x-1; i < N and j >= 0; i++, j--) mundo[i][j] = 1;
55          for(int i=y+1, j=x+1; i < N and j < M; i++, j++) mundo[i][j] = 1;
56      }
57      if (d == 'L')
58      {
59          FOR(i, x-1, -1) mundo[y][i] = 1;
60          for(int i=y-1, j=x-1; i >= 0 and j >= 0; i--, j--) mundo[i][j] = 1;
61          for(int i=y+1, j=x-1; i < N and j >= 0; i++, j--) mundo[i][j] = 1;
62      }
63      if (d == 'R')
64      {
65          For(i, x+1, M) mundo[y][i] = 1;
66          for(int i=y-1, j=x+1; i >= 0 and j < M; i--, j++) mundo[i][j] = 1;
67          for(int i=y+1, j=x+1; i < N and j < M; i++, j++) mundo[i][j] = 1;
68      }
69  }
70
71  int main()
72  {
73      int T;
74      cin>>T;
75      while(T--)
76      {
```

```
77          int F, E;
78          cin>>N>>M>>F>>E;
79
80          For(i, 0, N)
81              memset(mundo[i], 0, sizeof(mundo[i]));
82
83          For(i, 0, F)
84          {
85              char d;
86              int x, y;
87              cin>>y>>x>>d;
88              fuego(x, y, d);
89          }
90          For(i, 0, E)
91          {
92              char d;
93              int x, y;
94              cin>>y>>x>>d;
95              electrico(x, y, d);
96          }
97          bool escapar = false;
98          For(i, 0, N)
99              For(j, 0, M)
100                 if (!mundo[i][j])
101                     escapar = true;
102
103         if (!escapar)
104             cout<<"Escapar"<<endl;
105         else
106             cout<<"Pelear"<<endl;
107     }
108     return 0;
109 }
```

1.5. Problema E - Escribiendo Mensajes

Dificultad: ⋆
Temas: Implementación
Complejidad: $O(n)$

Entendiendo el problema

Dada una cadena de caracteres, se te pide encontrar el número de tecleos necesarios para escribir dicha cadena usando un teclado multitap.

Solución

Lo único que necesitamos es tener el número de tecleos, $T[x]$, necesarios para escribir el caracter x. Con esto podemos recorrer la cadena s e ir sumando $T[s[i]]$ a nuestro resultado.

Código

```cpp
#include <iostream>
#include <cstdio>
#define For(i, a, b) for(int i=a; i<b; ++i)
using namespace std;

int main()
{
    int T[26];
    for (int i=0, v=0; i<26; i++)
    {
        if (i+'a' == 's' or i+'a' == 'z')
            T[i] = 4;
        else
        {
            T[i] = v+1;
            v = (v+1)%3;
        }
    }

    int casos;
    string basura;
    cin>>casos;
    getline(cin, basura);
    For(j, 0, casos)
    {
        int total = 0;
        string L = "";
        getline(cin, L);
        For(i, 0, L.length())
            if (L[i] == ' ') total++;
            else total += T[L[i]-'a'];
        cout<<"Case #"<<j+1<<": "<<total<<endl;
    }
    return 0;
}
```

1.6. Problema F - ¿Y el problema F?

Dificultad: ⋆⋆
Temas: Cadenas, Implementación, Timewaster, Ordenamiento
Complejidad: $O(n \log n)$

Entendiendo el problema

Se te es dada una serie de cadenas y cada una de ellas está delimitada en subcadenas. Debes decidir cuál cadena es la cadena que tiene el mayor puntaje total; donde el puntaje total de cada cadena es la suma del puntaje de sus subacadenas, los puntajes para cada tipo de subacadena se definen en el problema. Una vez obtenida la cadena con mayor puntaje se deben imprimir, en orden lexicográfico, sus subcadenas que tienen un puntaje mayor a cero.

Solución

El problema anterior es trivial de resolver manteniendo un vector de vectores con las subcadenas útiles para cada cadena y al final del procesamiento se obtendría cuál es la que tuvo el mayor valor total de manera lineal para después solo ordenar sus subcadenas útiles, por lo que solo nos quedaría programar subrutinas que nos digan si una cadena es palíndroma regular (una cadena s es palíndroma si $s_i = s_{n-i-1}\ \forall\ i = 0, \ldots, n-1$), es espejo (una cadena s es espejo si $s_i = \text{mirror}(s_{n-i-1})\ \forall\ i = 0 \ldots n-1$, donde, mirror(c) es una función que regresa el caracter espejo de c) o ambas.

Por lo tanto la complejidad del problema sería $O(nm \log n)$, donde, n es la cantidad de cadenas y m es la longitud de la subcadena más grande de la cadena con el mayor valor.

Código

```
#include <iostream>
#include <algorithm>
#include <vector>
#include <map>
#define For(i, a, b) for(int i=a; i<b; ++i)
using namespace std;

bool isPalindromo(string A)
{
        int m1 = A.length()/2, m2 = A.length()/2;
        if (!(A.length()%2)) m1--;

        for(int i = m1, j = m2; i >= 0 and j < A.length(); i--, j++)
                if (A[i] != A[j]) return false;
        return true;
}

bool isEspejo(string A)
{
        char letra[] = {'A', 'E', 'H', 'I', 'J', 'L', 'M', 'O', 'S', 'T', 'U',
 ↪  'V', 'W', 'X', 'Y', 'Z', '1', '2', '3', '5', '8'};
        char espej[] = {'A', '3', 'H', 'I', 'L', 'J', 'M', 'O', '2', 'T', 'U',
 ↪  'V', 'W', 'X', 'Y', '5', '1', 'S', 'E', 'Z', '8'};
        map <char, char> letraEsp;
        For(i, 0, 21)
                letraEsp[letra[i]] = espej[i];

        int m1 = A.length()/2, m2 = A.length()/2;
        if (!(A.length()%2)) m1--;

        for(int i = m1, j = m2; i >= 0 and j < A.length(); i--, j++)
                if (A[j] != letraEsp[A[i]]) return false;
        return true;
}

bool isPalEsp(string A)
{
```

```
36          char letra[] = {'A', 'H', 'I', 'M', 'O', 'T', 'U', 'V', 'W', 'X', 'Y',
    ↪   '1', '8'};
37          bool sirve[26];
38          For(i, 0, 13)
39                  sirve[letra[i]-'A'] = true;
40
41          int m1 = A.length()/2, m2 = A.length()/2;
42          if (!(A.length()%2)) m1--;
43          for(int i = m1, j = m2; i >= 0 and j < A.length(); i--, j++)
44                  if (A[i] != A[j] or (A[i] == A[j] and !sirve[A[i]-'A'])) return
    ↪   false;
45          return true;
46  }
47
48  int main()
49  {
50          int T;
51          cin>>T;
52          while(T--)
53          {
54                  int M, P, valor[15] = {0}, nEsp[15] = {0};
55                  vector <vector <string> > utiles;
56                  string palabra[15];
57                  cin>>M>>P;
58                  utiles.assign(M, vector<string> ());
59                  For(i, 0, M)
60                  {
61                          int indice[60];
62                          cin>>palabra[i];
63                          For(k, 0, P)
64                                  cin>>indice[k];
65                          int pos = 0;
66                          For(k, 0, P)
67                          {
68                                  string A = "";
69                                  For(j, pos, indice[k]+1)
70                                          A = A + palabra[i][j];
71                                  //cout<<A<<endl;
72                                  pos = indice[k]+1;
73                                  if (isPalEsp(A))
74                                  {
75                                          valor[i] += 3;
76                                          utiles[i].push_back(A);
77                                          nEsp[i]++;
78                                  }
79                                  else if (isEspejo(A))
80                                  {
81                                          valor[i] += 2;
82                                          utiles[i].push_back(A);
83                                  }
84                                  else if (isPalindromo(A))
85                                  {
86                                          valor[i] += 1;
87                                          utiles[i].push_back(A);
88                                  }
89                          }
90                  }
```

```
91                  int mayor = 0;
92                  For(i, 0, M)
93                          if (valor[i] > valor[mayor])
94                                  mayor = i;
95                  sort(utiles[mayor].begin(), utiles[mayor].end());
96                  cout<<palabra[mayor]<<endl;
97                  cout<<nEsp[mayor]<<" "<<valor[mayor]<<endl;
98                  For(j, 0, utiles[mayor].size())
99                          cout<<utiles[mayor][j]<<endl;
100         }
101         return 0;
102 }
```

1.7. Problema G - Guiando a Mau por el Bosque

Dificultad: ⋆⋆⋆⋆
Temas: Teoría de Grafos, Programación Dinámica
Complejidad: $O(E + V \log V)$

Entendiendo el problema

Dado un grafo, encontrar la cantidad de rutas distintas que hay entre el origen s y el destino t con la restricción de que sólo se puede ir de un nodo u a un nodo v si la distancia mínima de v a t es menor que la distancia mínima de u a t.

Solución

Se puede observar que el grafo generado por sólo seguir dichas aristas es un grafo dirigido acíclico (DAG por sus siglas en inglés), que se puede generar fácilmente sacando todas las distancias desde t a todos los demás nodos con un algoritmo Dijkstra en $O(E + V \log V)$ y después con un pase a la lista de adyacencia formamos nuestro DAG.

Sobre un DAG la respuesta al número de caminos posibles desde un origen es bien conocido y se obtiene vistando los nodos en un orden topológico (DFS o BFS para obtener el orden $O(V + E)$) de manera que cuando visitemos un nodo ya habremos visitado todos los nodos que nos llevan a él usando la siguiente recurrencia:

$$f(s,u) = \begin{cases} 1 & u = s \\ \sum_{e=(v,u)\in E} f(s,v) & \text{cualquier otro caso} \end{cases}$$

done la línea de abajo nos dice que sumemos todas las aristas que llegan a u desde v, es decir, la cantidad de caminos que llegan a un nodo es la suma de la cantidad de caminos que llegan él.

Por último puesto que la respuesta puede ser muy grande usamos aritmética modular para presentar la solución.

Código

```cpp
#include <bits/stdc++.h>
using namespace std;
//look at my code my code is amazing
#define FOR(i, a, b) for (int i = int(a); i < int(b); i++)
#define FOREACH(it, a) for (typeof(a.begin()) it = (a).begin(); it != (a).end(); it++)
#define ROF(i, a, b) for (int i = int(a); i >= int(b); i--)
#define REP(i, a) for (int i = 0; i < int(a); i++)
#define INF 1000000000
#define INFLL 1000000000000000000LL
#define ALL(x) x.begin(), x.end()
#define MP(a, b) make_pair((a), (b))
#define X first
#define Y second
#define EPS 1e-9
#define DEBUG(x)   cerr << #x << ": " << x << " "
#define DEBUGLN(x) cerr << #x << ": " << x << " \n"
typedef pair<int, int> ii;
typedef vector<int> vi;
typedef long long ll;
typedef vector<bool> vb;
//ios_base::sync_with_stdio(0);//fast entrada/salida ;-)
//cin.tie(NULL); cout.tie(NULL);
int V, E;
vector< vector<pair<ll,ll> > > AdjList;
vector< vector<ll> > daglist;
vi dist, paths;
vb visited;
vi topo;

void toposort(ll u)
{
        visited[u] = true;
        REP(i, daglist[u].size())
        {
                ll v = daglist[u][i];
                if(not visited[v])
                {
                        toposort(v);
                }
        }
        topo.push_back(u);
}

void solve()
{
        AdjList.assign(V, vector<pair<ll,ll> >());
        daglist.assign(V, vector<ll>());
        visited.assign(V, false);
        topo.assign(0, 0);
        REP(i, E)
        {
                ll u, v, w;
                cin >> u >> v >> w;
                u--, v--;
                AdjList[u].push_back(pair<ll,ll> (v, w));
```

```
56                  AdjList[v].push_back(pair<ll,ll> (u, w));
57          }
58          set<ii> cola;
59          dist.assign(V, INF);
60          paths.assign(V, 0);
61          paths[0] = 1;
62          dist[1] = 0;
63          cola.insert(ii(0, 1));
64          while(not cola.empty())
65          {
66                  ll u = (*(cola.begin())).Y;
67                  ll d = (*(cola.begin())).X;
68                  cola.erase(cola.begin());
69                  if(d > dist[u])
70                          continue;
71                  REP(i, AdjList[u].size())
72                  {
73                          ll v = AdjList[u][i].X;
74                          ll w = AdjList[u][i].Y;
75                          if(d+w <= dist[v])
76                          {
77                                  dist[v] = d+w;
78                                  cola.insert(ii(dist[v], v));
79                          }
80                  }
81          }
82          REP(u, AdjList.size())
83          {
84                  REP(i, AdjList[u].size())
85                  {
86                          int v = AdjList[u][i].X;
87                          if(u != v and dist[v] < dist[u])
88                          {
89                                  daglist[u].push_back(v);
90                          }
91                  }
92          }
93          toposort(0);
94          ROF(i, topo.size()-1, 0)
95          {
96                  ll u = topo[i];
97                  REP(j, daglist[u].size())
98                  {
99                          ll v = daglist[u][j];
100                         paths[v] = (paths[v] + paths[u]) % 1000000009LL;
101                 }
102         }
103         cout << paths[1] << '\n';
104 }
105
106 int main()
107 {
108         ios_base::sync_with_stdio(0);//fast entrada/salida ;-)
109         cin.tie(NULL); cout.tie(NULL);
110         while(cin >> V and V)
111         {
112                 cin >> E;
```

```
113                 solve();
114         }
115         return 0;
116 }
```

1.8. Problema H - Haciendo Ananagramas

Dificultad: ⋆⋆
Temas: Ordenamiento, Cadenas
Complejidad: $O(n^2 * m \log m)$

Entendiendo el problema

Un ananagrama es una palabra que no es un anagrama, dado un diccionario debemos decir cuáles no son anagramas entre sí.

Solución

Una primera solución ingenua sería ver si cada palabra contra todas las demás es anagrama relativo y las que no lo son marcarlas para después imprimirlas, esto nos dejaría con el problema de ver si una palabra es anagrama de otra, esto se puede hacer ordenando los caracteres de ambas y si coinciden en todas sus letras estamos ante dos anagramas, esto nos deja ante una solución con complejidad $O(n^2 * m \log m)$ donde n es la cantidad de palabras y m es el tamaño de la palabra más grande, para este concurso este era la complejidad mínima esperada para pasar los casos de prueba del juez.

Un segudo enfoque es ordenar cada cadena e insertarlas en un árbol binario de búsqueda balanceado; para cada palabra checaremos si es que existe dentro de la estructura de datos así evitando la fuerza bruta cuadrática, esta solución tiene una complejidad de $O(n * m * \log(n) * \log(m))$.

Código

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

set<string> ananagrama;
vector<string> arr;

bool compare(string a, int apos, string b, int bpos)
{
    if(apos != bpos)
    {
        transform(a.begin(), a.end(), a.begin(), ::tolower);
        transform(b.begin(), b.end(), b.begin(), ::tolower);
        sort(a.begin(), a.end());
        sort(b.begin(), b.end());
        if(a == b)
            return true;
        else
```

```
18              return false;
19      }
20      return false;
21  }
22
23  int main()
24  {
25      string palabra;
26      while((cin >> palabra) and palabra != "#")
27      {
28          arr.push_back(palabra);
29      }
30      bool bandera=false;
31      for(int i=0; i<(int)arr.size(); ++i)
32      {
33          bandera = false;
34          for(int j=0; j<(int)arr.size(); ++j)
35          {
36              if ((bandera = compare(arr[i],i, arr[j],j)) and bandera)
37                  break;
38          }
39          if( not bandera)
40              ananagrama.insert(arr[i]);
41      }
42      for(set<string>::iterator  i=ananagrama.begin(); i!=ananagrama.end(); ++i)
43      {
44          cout << *i << "\n";
45      }
46      return 0;
47  }
```

Capítulo 2

Concurso de Programación de la XII Semana de MAC

2.1. Problema A - Asombroso League of Legends

Dificultad: $\star$
Temas: Ad-hoc
Complejidad: $O(1)$

Entendiendo el problema

Hay dos objetos que se mueven sobre una recta infinita, cuya posición al inicio es distinta una del otro, y quieres saber si el objeto a puede alcanzar al objeto b en algún momento en el tiempo; sabiendo que, el objeto a se mueve a una velocidad v_a durante c segundos y se mantiene inmóvil durante t segundos, mientras que el objeto b nunca para de moverse a una velocidad constans v_b.

Solución

Si observamos lo que pasa durante un ciclo de tiempo de tamaño $c + t$ podemos ver que el objeto a (el que persigue) solo se moverá $c * v_a$ unidades de distancia mientras que el objeto b se moverá $(c + t) * v_b$ unidades de distancia, por lo tanto si $c * v_a \leq (c + t) * v_b$ con cada ciclo de tiempo se irá alejando o manteniendo la misma distancia a b y en caso contrario b será alcanzado,sin importar donde empiecen.

Código

```
1 #include <iostream>
2 #include <cstdio>
3 using namespace std;
4
```

```
5  int main()
6  {
7  //        freopen("DataLoL.out","w",stdout);
8  //        freopen("DataLoL.in","r",stdin);
9          int n,c1,c2,T,C;
10         cin>>n;
11         while(n--)
12         {
13                 cin>>c1>>c2>>T>>C;
14                 c2 *= C;
15                 c1 = (C - T)*c1;
16                 if(c1 > c2)
17                 {
18                         cout<<"Se muere"<<endl;
19                 }
20                 else
21                 {
22                         cout<<"Se salva"<<endl;
23                 }
24         }
25 }
```

2.2. Problema B - Boby el Minotauro

Dificultad: $\star$
Temas: Ad-hoc, Cadenas
Complejidad: $O(n)$

Entendiendo el problema

El problema te pide saber si una cadena es un palíndromo.

Solución

Se dice que una cadena es un palíndromo si para elemento s_i de la cadena s se cumple que $s_i = s_{n-i+1}$ donde n es el tamaño de cadena e $i = 0, 1, \ldots, n-1$. Podemos checar esto eliminando los caracteres especiales y haciendo la comprobación lineal caracter por caracter, aunque con hacer la comprobación para una mitad es suficiente.

Código

```
1  #include <cstdio>
2  #include <iostream>
3  #include <cstdlib>
4  #include <cmath>
5  #include <algorithm>
6  #include <string>
7  #include <set>
8  #include <cctype>
9  #include <vector>
10 #include <map>
```

```cpp
#include <cctype>
using namespace std;
typedef long long ll;

string frase ;

void solve()
{
        int n = frase.size();
        string frase_bonita = "";

        for (int i = 0; i < n; ++i)
        {
                if(not (frase[i] == '.' or frase[i] == ',' or frase[i] == '!' or frase[i] == '?' or frase[i] == ' ' or frase[i] == '\n') )
                {
                //if(isalpha(frase[i]))
                        frase_bonita += frase[i];
                }
        }
        n = frase_bonita.size();
        for (int i = 0, j = n-1; i < (n/2)+1; ++i, --j)
        {
                if(frase_bonita[i] != frase_bonita[j])
                {
                        //cout << frase_bonita[i] << " " << frase_bonita[j] << '\n';
                        printf("OH NO!\n");
                        return;
                }
        }
        printf("NO SERAS COMIDO\n");
}

int main()
{
    //int k = 1;
        while(getline(cin, frase) and frase != "HECHO")
        {
                solve();
        }
        return 0;
}
```

2.3. Problema C - Coleccionista

Dificultad: ⋆
Temas: Implementación
Complejidad: $O(n)$

Entendiendo el problema

Dadas la cantidad de pokémon existentes, los pokémon que ya capturaste y los pokémon que no te interesan tener, debes obtener dos cosas:

- Cuántos y cuáles pokémon que sí te interesan te faltan obtener.
- Cuántos y cuáles pokémon de los que ya tienes no te interesan.

Solución

Para la solución necesitamos poder almacenar cuáles pokémon tenemos, y de esos cuáles no me interesan. Dado que el número total de pokémon es pequeño (≤ 1000), es posible crear arreglos de booleanos, `yatengo[]` y `nomeinteresan[]` para este propósito. Para responder la primer pregunta necesitamos contar los pokémon x tales que `yatengo[x] == false` y `nomeinteresan[x] == false`. Para la segunda, tenemos que contar aquellos pokémon tales que `yatengo[x] == true` y `nomeinteresan[x] == true`.

Código

```
#include<iostream>
#include<vector>
#include<cstdio>
using namespace std;
int main()
{
    int estampas,aux,tengo,desinteres,cont1,cont2,cont=0;;
    vector<int> yatengo;
    vector<int> nomeinteresan;
    while(cin>>estampas)
    {
        cont1=0;
        cont2=0;
        cont++;
        yatengo.assign(estampas+1,0);
        nomeinteresan.assign(estampas+1,0);
        cin>>tengo;
        for(int i=0;i<tengo;i++)
        {
            cin>>aux;
            yatengo[aux]=1;
        }
        cin>>desinteres;
        for(int i=0;i<desinteres;i++)
        {
            cin>>aux;
            nomeinteresan[aux]=1;
        }
        for(int i=1;i<estampas+1;i++)
        {
            if(nomeinteresan[i]==0 && yatengo[i]==0)
            {
                cont1++;
            }
        }
        cout<<cont1;
        for(int i=1;i<estampas+1;i++)
        {
```

```
39              if(nomeinteresan[i]==0 && yatengo[i]==0)
40              {
41                  cout<<" "<<i;
42              }
43          }
44          cout<<endl;
45          for(int i=1;i<estampas+1;i++)
46          {
47              if(nomeinteresan[i]==1 && yatengo[i]==1)
48              {
49                  cont2++;
50              }
51          }
52          cout<<cont2;
53          for(int i=1;i<estampas+1;i++)
54          {
55              if(nomeinteresan[i]==1 && yatengo[i]==1)
56              {
57                  cout<<" "<<i;
58              }
59          }
60          cout<<endl;
61      }
62      return 0;
63  }
```

2.4. Problema D - Domino

Dificultad: ⋆⋆
Temas: Búsquedas
Complejidad: $O(n \log n)$

Entendiendo el problema

Dado una línea alineada de dominos, debemos verificar si es posible hacer que la suma de las partes inferiores y la suma de las superiores es par, ésto mediante la rotación de las fichas de dominó.

Solución

Obtenemos las sumas de ambas partes, podemos dividir estas sumas en tres casos:

1. Si las sumas son pares, el problema esta resuelto y la respuesta es 0.

2. Si una suma es par y la otra impar no existe una solución debido a que cada ficha puede ser o con ambas partes pares o ambas impares o una par y la otra impar, pero al rotar cualquiera de estos tres tipos de fichas se sigue sin obtener una solución.

3. Si ambos sumas son impares, basta con encontrar una ficha con una parte par y la otra impar, al rotarla se obtendra una solución, si tal ficha no existe entonces no hay solución.

Código

```cpp
#include <iostream>
#include <sstream>
#include <algorithm>
#include <vector>
#include <map>
#include <set>
#include <queue>
#include <list>
#include <stack>
#include <cmath>
#include <cstdlib>
#include <cstdio>
#include <cstring>
#include <cctype>
#include <climits>
using namespace std;
int izq[105],der[105];
int n,sumi,sumd;

int main()
{
    int t,cont=0;
    cin>>t;
    while(t--)
    {
        cont++;
        cin>>n;
        cout<<"Caso "<<cont<<": ";
        sumi = sumd = 0;
        bool b = false;
        for(int i=0;i<n;i++)
        {
            cin>>izq[i]>>der[i];
            sumi += izq[i];
            sumd += der[i];
        }
        if((sumi&1) and (sumd&1))
        {
            for(int i=0; i<n; i++)
            {
                if( (izq[i]&1) != (der[i]&1))
                {
                    b = true;
                    break;
                }
            }
            if(b)
                cout<<1<<endl;
            else
                cout<<-1<<endl;
```

```
51          }
52          else if( !(sumi&1) and !(sumd&1) )
53          {
54              cout<<0<<endl;
55          }
56          else
57          {
58              cout<<-1<<endl;
59          }
60      }
61      return 0;
62  }
```

2.5. Problema E - Esto es fácil

Dificultad: ⋆⋆
Temas: Búsquedas
Complejidad: $O(n \log n)$

Entendiendo el problema

Dada una lista con n elementos, debemos determinar si existe un par de elementos de la lista cuya suma sea igual a un número M dado.

Solución

Para cada elemento a_j del arreglo A, mientras no hayamos determinado que existe una solucion, debemos verficar si existe a_k tal que $a_j + a_k = M$, eso lo realizamos mediante una busqueda binaria (debemos ordenar los elementos de A); esto debido a la limitante de tiempo del problema, si para cada elemento $a_j \in A$ buscaramos linealmente a su complemento a_k, la complejidad sería cuadrada y obtendríamos como veredicto *Tiempo Limite Excedido*.

Nota: Dado que los elementos de A son relativamente pequeños es posible disminuir la complejidad del problema a $O(n)$, esto marcado en un arreglo en la posición a_i si $a_i \in A$, ahorrandonos la el ordenamiento y la busqueda binaria.

Código

```
1   #include <iostream>
2   #include <cstdio>
3   #include <cstdlib>
4   #include <algorithm>
5   #define For(X,A,B) for(X=A; X<B; X++)
6   #define Maxt 1000
7   using namespace std;
8
9   int arr[1000]={0};
10  int var;
11
12  void busca(int a,int b,int valor)
13  {
```

```
14      if(a == b && arr[b]==valor)
15          {var = 1; return;}
16      else if(a == b && arr[b]!=valor)
17          {var = 0; return;}
18
19      int mitad = (a+b)>>1;
20
21      if(valor <= arr[mitad])
22          busca(a,mitad,valor);
23      else
24          busca(mitad+1,b,valor);
25  }
26
27  int main(int argc, char const *argv[])
28  {
29      //freopen("Data.in","r",stdin);
30      //freopen("Data.out","w",stdout);
31      int i,j,n,casos,sum,tam,x;
32
33      scanf("%d",&casos);
34      while(casos--)
35      {
36          scanf("%d",&tam);
37
38          for(i=0; i<tam; i++)
39              scanf("%d",&arr[i]);
40
41          scanf("%d",&sum);
42
43          //Cosas
44          var = 0;
45          sort(arr,arr+tam);
46          For(j,0,tam)
47          {
48              x = sum - arr[j];
49              if(x>0)
50                  busca(0,tam-1,x);
51
52              if(var)
53              {
54                  printf("SI\n");
55                  break;
56              }
57          }
58          if(var == 0)
59              printf("NO\n");
60      }
61      return 0;
62  }
```

2.6. Problema F - Falsa Simulación

Dificultad: ⋆⋆
Temas: Ad-Hoc, Estructuras de Datos
Complejidad: $O(n+m)$ construcción, $O(1)$ por consulta.

Entendiendo el problema

Para el propósito de entender el problema supongamos que se tiene una matriz de n filas por m columnas de la siguiente manera, suponiendo que $n = m = 5$:

$$\begin{array}{c} \\ r_1 \\ r_2 \\ r_3 \\ r_4 \\ r_5 \end{array} \begin{array}{c} \begin{array}{ccccc} c_1 & c_2 & c_3 & c_4 & c_5 \end{array} \\ \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 \\ 16 & 17 & 18 & 19 & 20 \\ 21 & 22 & 23 & 24 & 25 \end{pmatrix} \end{array}$$

Si quisieramos efectuar la operación `R 1 2` obtendriamos lo siguiente:

$$\begin{array}{c} \\ r_2 \\ r_1 \\ r_3 \\ r_4 \\ r_5 \end{array} \begin{array}{c} \begin{array}{ccccc} c_1 & c_2 & c_3 & c_4 & c_5 \end{array} \\ \begin{pmatrix} 6 & 7 & 8 & 9 & 10 \\ 1 & 2 & 3 & 4 & 5 \\ 11 & 12 & 13 & 14 & 15 \\ 16 & 17 & 18 & 19 & 20 \\ 21 & 22 & 23 & 24 & 25 \end{pmatrix} \end{array}$$

de donde la respuesta para `W 9` sería `1 4`.

Solución

La solución ingenua consiste en mantener una matriz e ir simulando cada uno de las consultas en $O(n)$ u $O(m)$, pero se puede ver en primera instancia que esta solución no es factible para nuestras restricciones ya que ni siquiera pasaría las restricciones de memoria.

Una solución que nos permite hacer consultas y modificaciones a la matriz en $O(1)$, se logra mediante el uso de dos vectores por renglones y columna uno que te relacione el índice original y índice actual, y visceversa.

Sólo queda saber que para obtener los índices de una matriz de la forma original dado el valor de z debemos hacer: $x = \lfloor z/m \rfloor + 1$, $y = (z \mod m) + 1$; y para obtener z conociendo x y y hacemos: $z = (x-1)m + y$.

Código

```
1 #include <iostream>
2 #include <cstdio>
3 #include <algorithm>
```

```
4  #include <vector>
5  using namespace std;
6  typedef vector<int> vi;
7
8  #define N 1234
9  #define M 5678
10
11 vi rows(N+3, 0);
12 vi cols(M+3, 0);
13 vi rr = rows;
14 vi cc = cols;
15
16 void solveR()
17 {
18         int x, y;
19         scanf("%d %d", &x, &y);
20         swap( rows[x], rows[y] );
21         rr[rows[x]] = x;
22         rr[rows[y]] = y;
23 }
24
25 void solveC()
26 {
27         int x, y;
28         scanf("%d %d", &x, &y);
29         swap( cols[x], cols[y] );
30         cc[cols[x]] = x;
31         cc[cols[y]] = y;
32 }
33
34 void solveQ()
35 {
36         int x, y;
37         scanf("%d %d", &x, &y);
38         printf("%d\n", ((rows[x]-1) * M) + cols[y] );
39 }
40
41 void solveW()
42 {
43         int z;
44         cin >> z;
45         --z;
46         printf("%d %d\n", rr[(z / M)+1], cc[(z%M)+1]);
47 }
48
49 int main()
50 {
51         for (int i = 0; i < M+3; ++i)
52         {
53                 cc[i] = cols[i] = i;
54         }
55         for (int i = 0; i < N+3; ++i)
56         {
57                 rr[i] = rows[i] = i;
58         }
59         char orden;
60         while(cin >> orden)
```

```
61          {
62                  switch(orden)
63                  {
64                          case 'R':
65                                  solveR();
66                                  break;
67                          case 'C':
68                                  solveC();
69                                  break;
70                          case 'Q':
71                                  solveQ();
72                                  break;
73                          case 'W':
74                                  solveW();
75                                  break;
76                  }
77          }
78          return 0;
79  }
```

2.7. Problema G - Geómetra Hermann

Dificultad: $\star$
Temas: Matemáticas
Complejidad: $O(1)$

Entendiendo el problema

Debes encontrar el área de un círuclo con radio r utilizando una distancia euclideana y un círculo con radio r utilizando una distancia Manhattan.

Solución

El círculo en una geometría con la distancia euclideana es trivial. Para el otro caso hay que observar que un círculo en una geometría con la distancia Manhattan es en realidad un cuadrado rotado de lado $\sqrt{2r^2}$, por lo tanto su área es $2r^2$.

Código

```
1   #include <stdio.h>
2   #include <math.h>
3
4   int main()
5   {
6       double PI = atan(1.0)*4.0;
7       double R;
8       while(scanf("%lf", &R)==1)
9       {
10          printf("%.4lf\n", (double)PI*R*R);
11          printf("%.4lf\n", (double)R*R*2.0);
12      }
```

```
13      return 0;
14  }
```

2.8. Problema H - Historia de los relojes

Dificultad: ⋆
Temas: Matemáticas
Complejidad: $O(1)$

Entendiendo el problema

Dado un ángulo, quieres saber si es posible que las dos manecillas de un reloj convencional, con 60 marcas, pueden crear dicho ángulo.

Solución

Dado que sólo hay 60 marcas en el reloj, los 360 grados estarán divididos en 60 marcas separadas por 6 grados, así que las manecillas podrán formar un ángulo si y sólo si el ángulo es múltiplo de 6.

Código

```
#include<iostream>
#include<cstdlib>
#include<ctime>
#include<cstdio>
using namespace std;
int main()
{
    int a,cont=0;
    while(cin>>a)
    {
        cont++;
        cout<<"Caso "<<cont<<": ";
        if(a%6)
        {
            cout<<"N\n";
        }
        else
        {
            cout<<"Y\n";
        }
    }
}
```

2.9. Problema I - Investigando laberintos

Dificultad: ⋆⋆
Temas: Flood fill
Complejidad: $O(nm)$

Entendiendo el problema

Dado una cuadrícula de $n \times m$, con algunas paredes dentro de ella, y un punto inicial, se te pide marcar todos los lugares dentro de la cuadrícula a los que puedes llegar, sabiendo que no puedes atravezar paredes y que sólo te puedes mover hacia arriba, abajo, izquierda o derecha.

Solución

Este es un problema muy común y su solución es conocida como algoritmo de relleno por difusión (flood fill en inglés), que es básicamente una búsqueda.

Código

```
#include<iostream>
#include<string>
#include<vector>
#include<cstdio>
using namespace std;
vector<string> v;
int dfs(int i,int j)
{
    if(i>0 && j>0 && j<v[i].size() && i <v.size() && v[i][j]!='#' && (v[i][j]==' ' || v[i][j]=='*'))
    {
        v[i][j]='#';
        dfs(i,j+1);
        dfs(i+1,j);
        dfs(i-1,j);
        dfs(i,j-1);
    }
    return 0;
}
int main()
{
    string aux;
    int t,a,b;
    cin>>t;
    while(t--)
    {
        cin>>a>>b;//a=filas, b=columnas
        getline(cin,aux);
        for(int i=0;i<a;i++)
        {
            getline(cin,aux);
            v.push_back(aux);
        }
        for(int k=0;k<v.size();k++)
        {
            for(int l=0;l<v[k].size();l++)
            {
                if(v[k][l]=='*')
                {
                    dfs(k,l);
                }
```

```
41              }
42          }
43          for(int k=0;k<v.size();k++)
44          {
45              cout<<v[k]<<endl;
46          }
47          cout<<endl;
48          v.clear();
49      }
50      return 0;
51  }
```

Capítulo 3

4.º Concurso Interno de Programación

3.1. Problema A - Annie la Hija de la Obscuridad

Dificultad: ⋆
Temas: Ad-hoc
Complejidad: $O(1)$

Entendiendo el problema

Dadas las posiciones de Annie y de un campeón en el plano, se nos pide saber si Annie es capaz de encadenar los tres hechizos que conoce al campeón enemigo. El problema nos dice que no habrá obstáculos entre ellos y que el lanzamiento de los hechizos es de manera instantánea.

Solución

Lo que necesitamos para poder encadenar los tres hechizos es que la distancia entre Annie y el campeón sea menor o igual a $500u$ ya que de éste modo los tres ataques pueden alcanzar al campeón.

Código

```
1  #include<bits/stdc++.h>
2
3  using namespace std;
4
5  #define FOR(i, a, b) for (int i = int(a); i < int(b); i++)
6  #define REP(i, a) for (int i = 0; i < int(a); i++)
7  #define INF 1000000000
8  typedef pair<int, int> ii;
```

```cpp
typedef vector<int> vi;
typedef vector<ii> vii;
typedef long long ll;
typedef vector<bool> vb;

int casos, q;
ii annie, campeon;

int distancia_squared(ii u, ii v)
{
    return (u.first-v.first)*(u.first-v.first) +
    ↪  (u.second-v.second)*(u.second-v.second);
}

void querea()
{
    scanf("%d %d", &campeon.first, &campeon.second);
    int dis_squ = distancia_squared(annie, campeon);
    if( dis_squ <= 250000 )
        printf("FULL COMBO\n");
    else
        printf("OUTPLAYED\n");

}

void solve()
{
    scanf("%d %d", &annie.first, &annie.second);
    scanf("%d", &q);
    REP(i, q)
        querea();
}

int main()
{
    freopen("Annie.in", "r", stdin);
    freopen("Annie.out", "w", stdout);
    scanf("%d", &casos);
    while(casos--)
        solve();
    return 0;
}
```

3.2. Problema B - Colores

Dificultad: ⋆
Temas: Ad-hoc
Complejidad: $O(1)$

Entendiendo el problema

Nos dan un diccionario de colores cuyo código está en binario. En el problema se te da el código alterado (intercambiando unos y ceros) de un color y se te pide imprimir el color correspondiente del código original.

Solución

Directa.

Código

```cpp
#include<iostream>

using namespace std;

typedef vector<int> vi;
typedef vector<pair<int,int> > vii;
typedef vector<vi> vvi;
#define rep(a,b) for(int a=0;a<b;a++)
#define For(a,b,c) for(int a=b;a<c;a++)
int main()
{
    std::ios_base::sync_with_stdio(false);
    int n,a,b,c;
    string v[8] =
    {"Negro","Naranja","Verde","Amarillo","Morado","Rojo","Azul","Blanco"};
    cin>>n;
    rep(i,n){
        cin>>a>>b>>c;
        cout<<v[4*a + 2*b + c]<<endl;
    }
    return 0;
}
```

3.3. Problema C - CandyLand

Dificultad: ⋆
Temas: Sliding window
Complejidad: $O(n)$

Entendiendo el problema

El problema se reduce a: dado un arreglo de n enteros, encontrar el subarreglo de tamaño exactamente k, tal que la suma del subarreglo sea máxima.

Solución

La solución $O(nk)$ es demasiado lenta para este problema. Sea `a[]` el arreglo de enteros. Si declaramos un arreglo `v[i]` $:= \sum_{j=1}^{i}$ `a[j]`, entonces la suma del subarreglo de tamaño k empezando en la posición i es: `v[i+k]-v[i-1]`. Con esta observación tenemos una solución $O(n)$.

Código

```cpp
#include <iostream>
#include <cstdio>
```

```cpp
#include <cstring>
#include <algorithm>

#define For(i, a, b) for(int i=a; i<b; ++i)
#define INF (1<<30)

using namespace std;

int v[1000005];

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

    while (T--)
    {
        int N, K;
        scanf("%d %d", &N, &K);

        int res = -INF, ind = 1;
        For(i, 1, N+1)
        {
            scanf("%d", &v[i]);
            v[i] += v[i-1];
        }

        For(i, K, N+1)
            if (v[i] - v[i-K] > res)
                res = v[i]-v[i-K], ind = i-K+1;

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

    return 0;
}
```

3.4. Problema D - Raíces digitales

Dificultad: ⋆⋆
Temas: Ad-hoc
Complejidad: $O(k)$

Entendiendo el problema

En el problema se define $S(n)$ como la suma de los dígitos de n y

$$rd(n) = \begin{cases} S(n) & \text{si } S(n) < 10 \\ rd(S(n)) & \text{si } S(n) \geq 10 \end{cases}$$

Se nos da $1 \leq k \leq 100$ y $1 \leq d \leq 9$, y se nos pide encontrar el número más pequeño min y el número más grande max tales que $rd(min) = rd(max) = d$ y tanto min como max contengan k dígitos.

Solución

Éstos dos números se construyen de la siguiente manera:

$$min = (1)(\underbrace{0 \ ... \ 0}_{k-2})(d-1)$$
$$max = (\underbrace{9 \ ... \ 9}_{k-1})(d)$$

Código

```java
import java.util.*;
import java.io.*;

class D
{
    public static void main(String[] args)
    {
        MyScanner sc = new MyScanner();
        int T = sc.nextInt();

        while (T-- > 0)
        {
            int k = sc.nextInt(), d = sc.nextInt();

            for(int i = 0; i < k-1; i++)
                System.out.print(9);
            System.out.println(d);

            if (k > 1)
                System.out.print(1);
            for(int i = 1; i < k-1; i++)
                System.out.print(0);
            if (k > 1)
                System.out.println(d-1);
            else
                System.out.println(d);
        }
    }
};
```

3.5. Problema E - Buscando asiento

Dificultad: ⋆⋆
Temas: Ad-hoc
Complejidad: $O(1)$

Entendiendo el problema

El problema nos dice que hay A niños que se quieren sentar en B asientos, sin embargo cada uno de los niños quiere sentarse en un asiento tal que sus dos vecinos estén desocupados. Los niños van sentándose uno por uno escogiendo

cualquier asiento que cumpla con los requerimientos. Debemos ver si siempre es posible que estén agusto independientemente de cómo se vayan sentado (caso "Si"), si depende de los asientos que escojan (caso "Tal vez"), o sin importar que asientos escojan no es posible que estén agustos (caso "No").

Solución

1. Caso "No". Si $A > \left\lceil \frac{B}{2} \right\rceil$. Pues la mejor forma en que se pueden ir sentando es dejando un espacio de separación, si así no es posible sentarlos, entonces no hay forma de que estén agusto.

2. Caso "Si". Si $A \leq \left\lceil \frac{B}{3} \right\rceil$. La "peor" forma en la que se pueden acomodar los niños es si cada uno se coloca a dos asientos de separación de sus vecinos, ya que si se sientan a un asiento de separación sería la forma más "compacta", y si se sientan a más de dos, entonces un tercer niño puede sentarse en medio de ellos. Ésto se vería de la siguiente forma:

$$..,010\underbrace{010}_{3}010...$$

   Por lo tanto se puede apreciar que cada niño "cubre" tres asientos.

3. Caso "Tal vez". Si $\left\lceil \frac{B}{3} \right\rceil < A \leq \left\lceil \frac{B}{2} \right\rceil$

Código

```java
import java.util.Scanner;

class E
{
    public static void main(String[] args)
    {
        Scanner sc = new Scanner(System.in);

        int T = sc.nextInt();

        while (T-- > 0)
        {
            int n = sc.nextInt(), s = sc.nextInt();

            int limNo = s % 2 == 0 ? s / 2 : s / 2 + 1;
            int limSi = s % 3 == 0 ? s / 3 : s / 3 + 1;

            if (n > limNo)
                System.out.println("No");
            else if (n <= limSi)
                System.out.println("Si");
            else
                System.out.println("Tal vez");
        }
    }
};
```

3.6. Problema F - Fibonacci

Dificultad: ⋆
Temas: Ad-hoc
Complejidad: $O(1)$

Entendiendo el problema

En este caso se nos da una cadena de dígitos F cuya construcción se parece a la sucesión de Fibonacci. La forma de construirla es sumar los dos últimos dígitos de la cadena y concatenar el resultado. Los primeros dígitos de la cadena son los siguientes:

$$\underbrace{1123581347}_{10}112...$$

Y se nos pide saber cuál es el n-ésimo dígito de la cadena F.

Solución

Los dígitos señalados siempre se van a repetir, debido a que se toman los dos últimos dígitos para construir los siguientes. Por lo tanto sólo debemos guardar los diez primeros dígitos y el n-ésimo dígito de F será $F[n\,\%10]$.

Código

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

using namespace std;
typedef vector<int> vi;
typedef vector<pair<int,int> > vii;
typedef vector<vi> vvi;
#define rep(a,b) for(int a=0;a<b;a++)
#define For(a,b,c) for(int a=b;a<c;a++)
int main()
{
    std::ios_base::sync_with_stdio(false);
    int n,x,v[10]={1,1,2,3,5,8,1,3,4,7};//1123581347
    cin>>n;
    while(n--)
    {
        cin>>x;
        x--;
        printf("%d\n",v[x%10]);
    }
    return 0;
}
```

3.7. Problema G - Código

Dificultad: ⋆
Temas: Ad-hoc
Complejidad: $O(NM)$

Entendiendo el problema

En el problema se nos da una S cadena de tamaño $N \times M$ codificada. La forma en que se codifico la cadena fue metiéndola en una matriz de tamaño $N \times M$ y rotarla 90 grados en sentido del reloj. Después de ésto se agarran los caracteres dentro de la matriz rotada en orden de arriba hacia abajo y de izquierda a derecha. La tarea consiste en decodificar el mensaje.

Solución

Para hacerlo hay que meter la cadena en una matriz de $M \times N$, rotarla 90 grados en sentido contrario al reloj e imprimir los caracteres en el mismo orden. Esto se puede lograr con puro manejo de índices.

Código

```
#include<iostream>
#include<cstdio>
#include<cstdlib>
#include<string>
using namespace std;
int main()
{
    int t,m,n;
    string s;
    cin>>t;
    while(t--)
    {
        cin>>m>>n;
        cin>>s;
        for(int i=m-1;i>=0;i--)
            for(int j=0;j<n;j++)
                cout<<s[i+m*j];
        cout<<endl;
    }

    return 0;
}
```

Capítulo 4

Concurso de Programación de la XIII Semana de MAC

4.1. Problema A - Avería

Dificultad: $\star$
Temas: Matemáticas
Complejidad: $O(\log \text{mín}(a, b))$

Entendiendo el problema

Se tienen dos limpiaparabrisas, los cuales realizan un ciclo en a y b segundos, respectivamente. Se dice que los limpiaparabrisas empiezan a funcionar al mismo tiempo, en la misma posición, y se quiere saber en cuánto tiempo volveran en encontrarse en la misma posición por primera vez.

Solución

Lo que buscamos son dos números k, q tal que $ak = bq$, y que ak sea mínimo. La respuesta es ak. Es fácil ver que $ak = bq = lcm(a, b)$ nos minimiza este valor.

Caso Fácil

Para el caso fácil, se puede ir aumentando por si misma una de las variables hasta que la otra la divida exactamente, esto es el primer momento en el que ambos se encuentran en su punto de inicio otra vez.

Caso Difícil

Para el caso difícil, las restricciones no nos permiten hacer el mismo cálculo debido al numero de casos de prueba.

La solución es ver que el Mínimo Común Multiplo de dos números se puede calcular de la siguiente forma:

$$\text{MCM}(a,b) = \frac{ab}{\text{MCD}(a,b)}$$

donde MCD es el máximo común divisor de los dos números.

Ahora, para calcular el Máximo Común Divisor de dos números eficientemente, podemos utilizar el algoritmo de Euclides.

La siguiente rutina basada en ese algoritmo nos devuelve el MCD(a,b) y la calcula en $O(\log n)$ donde n es el máximo de a y b.

Código

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

int gcd(int a, int b) { return b == 0 ? a : gcd(b, a % b); }
int lcm(int a, int b) { return a * (b / gcd(a, b)); }

int main()
{
    int n, a, b;
    cin >> n;
    for(int i = 0; i < n; i++)
    {
        cin >> a >> b;
        cout << lcm(a, b) << endl;
    }
    return 0;
}
```

4.2. Problema B - Bob y Peter

Dificultad: ⋆⋆
Temas: Cadenas
Complejidad: $O(n)$

Entendiendo el problema

Dada una cadena tenías que decir si es un palíndromo (caso fácil), o tiene una subsecuencia que sí es palíndromo (caso difícil).

Solución

Solución Fácil

Aquí sólo tenías que revisar las cadenas eran palíndromos o no, entonces el algoritmo sería más o menos así:

```
N = Numero de casos
for(i = 1; i <= N; i++)
       P_i = palabra i
    Espalindromo = 1
    for(pos=0; pos<tam(P_i); pos++)
        if(P_i[pos] != P_i[size(P_i) - 1 - i]
               Espalindromo = 0;
    if (Espalindromo == 1)
        Imprime(SI)
    else
        Imprime(NO)
```

Solución (difícil)

La solución más sencilla de programar de este algoritmo era revisando todos los tamaños, pero ésto es muy tardado de calcular para una computadora, ya que para cada palabra se tienen máximo 1000 letras, y revisar que exista algún palíndromo de tamaño 2 hasta 1000 requiere $\approx 10^6$ cálculos (≈ 1000 comparaciones por cada tamaño de palíndromo), y eso multiplicado por las 2000 palabras, da un total de $\approx 2 * 10^9$ cálculos, lo cuál, por lo general, le toma más de 1 segundo hacerlos a una computadora común.

Pero si analizamos un poco los palíndromos, todos los palíndromos de tamaño ≥ 2 tienen en el centro de estos un subpalíndromo de tamaño 2 o 3, por lo que sólo buscaremos palíndromos de tamaño 2 o palíndromos de tamaño 3 (si no encontramos algún palíndromo de tamaño 2, menos encontraremos un palíndromo de tamaño 4, 6, 8...; y de manera similar para los palíndromos de tamaño impar).

```
T = Numero de casos
for(i = 1; i <= T; i++)
       P_i = palabra i
    TieneSubcad = 0
    for(pos = 1; pos < tam(P_i); pos++)
        if(P_i[pos] == P_i[pos-1])
               TieneSubcad = 1 //Palindromo de tamanio 2
    for(pos = 2; pos < tam(P_i); pos++)
        if(P_i[pos] == P_i[pos-2])
               TieneSubcad = 1 //Palindromo de tamanio 3
    if(TieneSubcad == 1)
        Imprime(SI)
    else
        Imprime(NO)
```

4.3. Problema C - Conejos

Dificultad: ⋆⋆⋆⋆
Temas: Matemáticas
Exponenciación de matrices
Complejidad: $O(\log n)$

Entendiendo el problema

El problema consistía en encontrar el n-ésimo fibonacci par.

Solución

Sea $f_n = 4f_{n-1} + f_{n-2}$, con $f_1 = 2$ y $f_2 = 8$. Mostraremos que f_n nos da el n-ésimo fibonacci par. Dado que cada tercer elemento de la sucesión de fibonacci es par, basta con probar que $fib_n = 4fib_{n-3} + fib_{n-6}$.

$$\begin{aligned} fib(n) &= fib_{n-1} + fib_{n-2} \\ &= fib_{n-2} + 2fib_{n-3} + fib_{n-4} \\ &= 3fib_{n-3} + 2fib_{n-4} \\ &= 3fib_{n-3} + fib_{n-4} + fib_{n-5} + fib_{n-6} \\ &= 4fib_{n-3} + fib_{n-6} \end{aligned}$$

Sea $f_n = 4f_{n-1} + f_{n-2}$, con $f_1 = 2$ y $f_2 = 8$. Mostraremos que f_n nos da el n-ésimo fibonacci par. Dado que cada tercer elemento de la sucesión de fibonacci es par, basta con probar que $fib_n = 4fib_{n-3} + fib_{n-6}$.

$$\begin{aligned} fib(n) &= fib_{n-1} + fib_{n-2} \\ &= fib_{n-2} + 2fib_{n-3} + fib_{n-4} \\ &= 3fib_{n-3} + 2fib_{n-4} \\ &= 3fib_{n-3} + fib_{n-4} + fib_{n-5} + fib_{n-6} \\ &= 4fib_{n-3} + fib_{n-6} \end{aligned}$$

Con esto sólo necesitamos calcular f_n.

Caso Fácil

Para el caso fácil era posible hacerlo en $O(n)$ con un **for**.

Caso Difícil

Sin embargo para las restricciones difíciles ésto ya no era posible. Para resolver el caso difícil necesitamos observar que:

$$\begin{pmatrix} f_3 \\ f_2 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} f_2 \\ f_1 \end{pmatrix}$$

Utilizando el mismo razonamiento tenemos:

$$\begin{aligned}\begin{pmatrix} f_4 \\ f_3 \end{pmatrix} &= \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} f_3 \\ f_2 \end{pmatrix} \\ &= \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^2 \begin{pmatrix} f_2 \\ f_1 \end{pmatrix}\end{aligned}$$

Generalizando ésta idea se obtiene:

$$\begin{pmatrix} f_n \\ f_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{n-2} \begin{pmatrix} f_2 \\ f_1 \end{pmatrix}$$

Ahora sólo necesitamos obtener $\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n$ de forma eficiente. Basándonos en el hecho de que

$$x^n = \begin{cases} (x^{\frac{n}{2}})(x^{\frac{n}{2}}) & \text{si } n \text{ es par} \\ x(x^{\frac{n}{2}})(x^{\frac{n}{2}}) & \text{si } n \text{ es impar} \end{cases}$$

podemos utilizar el siguiente procedimiento para calcularlo en $O(\log(n))$

Código

```
#include <iostream>
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <vector>
#include <bitset>
#include <cmath>
#include <queue>

#define For(i, a, b) for(int i=(a); i<(b); ++i)
#define INF (1<<30)
#define MP make_pair
#define MOD 1000000007

using namespace std;

typedef pair <int, int> ii;
typedef vector <vector <int> > vvi;

int madd(int a, int b)
{
    return (a + b) % MOD;
}

int mmult(int a, int b)
{
    return (1LL*a*b) % MOD;
}

```

```
30  vvi matmult(vvi A, vvi B)
31  {
32      vvi C(2, vector <int>(2, 0));
33      For(i, 0, 2)
34          For(j, 0, 2)
35              For(k, 0, 2)
36                  C[i][j] = madd( C[i][j], mmult(A[i][k], B[k][j]) );
37
38      return C;
39  }
40
41  vvi matexp(vvi X, long long n)
42  {
43      if (n == 1)
44          return X;
45
46      vvi ans = matexp(X, n >> 1);
47      if (n & 1)
48          return matmult( matmult(ans, ans), X );
49
50      return matmult(ans, ans);
51  }
52
53  int main()
54  {
55      int tt;
56      scanf("%d", &tt);
57
58      while (tt--)
59      {
60          long long n;
61          scanf("%lld", &n);
62
63          if (n == 1)
64              printf("2");
65          else if (n == 2)
66              printf("8");
67          else
68          {
69              vvi X(2, vector <int> (2, 1));
70              X[0][0] = 4;
71              X[1][1] = 0;
72
73              X = matexp(X, n-2);
74
75              printf("%d\n", madd( mmult(X[0][0], 8), mmult(X[0][1], 2) ) );
76          }
77      }
78
79      return 0;
80  }
```

4.4. Problema D - Desafío

Dificultad: $\star$
Temas: Matemáticas
Complejidad: $O(1)$

Entendiendo el problema

Dado un entero N se quiere determinar la suma de los múltipos de 3 y 5 que son menores a N. No se deben sumar dos o más veces el mismo número.

Solución

Caso Fácil

Para el caso fácil podemos correr 2 ciclos que verifiquen qué números son divisibles por 3 y por 5 respectivamente, sumarlos, incrementar `i` en 3, 5, y en alguno de los 2 ciclos no contar a aquellos números que son múltiplos de ambos, ya que estaríamos contándolos 2 veces. Esta idea corre en $O(n)$. Lo cual es bueno si n no es "grande".

Caso Difícil

Para el caso dificil, la idea anterior no funcionaría ya que las restricciones obligarían a usar un ciclo demasiado grande, entonces usamos una idea mejor:

Pensemos en la suma de los primeros n múltiplos de 3 como:

$$3 + 6 + 9 + \ldots + 3n$$
$$3(1 + 2 + 3 + \ldots + n)$$

Nos percatamos que el segundo factor de la expresión anterior es la suma de los primeros n naturales, entonces queda como:

$$3\frac{n(n+1)}{2}$$

De manera similar para el caso de 5:

$$5\frac{n(n+1)}{2}$$

Entonces la suma de las expresiones anteriores es casi lo que queremos, esto porque en cada una de ellas se está contemplando los múltiplos de 5 y 3 al mismo tiempo, así que solo necesitamos eliminar una vez la suma de éstos:

$$15\frac{n(n+1)}{2}$$

Ahora ¿cómo encontramos los primeros n múltiplos menores a N?, fácil, basta con hacer una división entera de $N-1$ entre el divisor deseado. De esta manera nuestro programa se vería de la siguiente forma:

Código

```
#include <iostream>
#include <cstdio>
#define For(x,a,b) for(int x=a; x<b; x++)
using namespace std;

typedef long long ll;

ll mtres(ll n){ll r = n/3; return (r*(r+1)/2)*3;}
ll mcinco(ll n){ll r= n/5; return (r*(r+1)/2)*5;}
ll mquince(ll n){ll r= n/15; return (r*(r+1)/2)*15;}

int main()
{
        int T,N;

        scanf("%d",&T);

        For(j,0,T)
        {
                scanf("%d",&N);

                printf("%lld\n",mtres(N-1)+mcinco(N-1)-mquince(N-1));
        }
        return false;
}
```

4.5. Problema E - Embriagándose con Gragas

Dificultad: ⋆
Temas: Ad-Hoc
Complejidad: $O(m)$

Entendiendo el problema

Empezaré por explicar un poco más el problema, tomemos como referencia el caso de entrada ejemplo:

```
5 3
1 2 100
2 5 100
3 4 100
```

Dice que tenemos 5 barriles inicialmente vacíos:

```
0 0 0 0 0
```

Después de la primera operación que consta de añadir 100 litros de cerveza a los barriles que van desde 1 hasta 2 inclusive, tenemos:

```
100 100 0 0 0
```

Para la segunda operación:

```
100 200 100 100 100
```

Por último la tercera operación

```
100 200 200 200 100
```

Por lo que tenemos un total de 800 litros litros litros de cerveza distribuidos en 5 barriles, lo que nos da un promedio de $\frac{800}{5} = 160$.

Solución

Caso Fácil

Dicho lo anterior la primer solución que se nos puede ocurrir es simular las operaciones conservando un vector de tamaño n donde iremos almacenando las sumas sobre cada barril y al final obtenemos el promedio de todo lo dado.

El problema con la solución anterior es que su peor caso se da cuando para cada operación (m en total) nos piden llenar los barriles con índice desde 1 hasta n, lo cual tiene un complejidad de $O(nm)$ que es lo suficientemente buena para el caso fácil, pero que jamás pasará en tiempo el caso difícil.

Caso Difícil

Para el caso difícil basta ver que en cada operación el total de cerveza en todos los barriles aumentará la cantidad en $(b - a + 1) * k$ litros haciendo el cálculo de cada operación constante en tiempo, esto es por que se aumentarán k litros por cada barril en el rango de a hasta b, por lo tanto, el tiempo total en el peor caso será $O(m)$ (más que suficiente para pasar ambos casos) y solo quedará por ver que el resultado puede ser muy grande y habrá que almacearlo en una variable entera de 64 bits (`long long`).

El siguiente código muesta una función que calcula cada caso el resultado incluyendo la lectura y salida de datos.

Código

```
1  void solve()
2  {
3          long long n, m;
4
5          cin >> n >> m;
6
7          long long total = 0, a, b, k;
8
9          for(int i=0; i<m; i++)
10         {
11                 cin >> a >> b >> k;
12                 total += (b-a+1)*k;
13         }
```

```
14
15          cout << total / n << "\n";
16  }
```

Capítulo 5

5.º Concurso Interno de Programación

5.1. Problema A - Analizando Exámenes

Dificultad: $\star$
Temas: Ad-hoc
Complejidad: $O(n)$

Entendiendo el problema

Dados dos arreglos A y B, de tamaño n, se te pide saber cuántos índices i cumplen que $a[i] = b[i]$.

Solución

Directa.

Código

```
1  import java.util.*;
2
3  class Amain
4  {
5      public static void main(String[] args)
6      {
7          Scanner sc = new Scanner(System.in);
8
9          int a[] = new int[1005], b[] = new int[1005];
10         int n = sc.nextInt();
11
12         for(int i = 0; i < n; i++)
13             a[i] = sc.nextInt();
14
15         for(int i = 0; i < n; i++)
```

```
16              b[i] = sc.nextInt();
17
18          int ans = 0;
19          for(int i = 0; i < n; i++)
20              ans += a[i] == b[i] ? 1 : 0;
21
22          System.out.println(ans);
23      }
24  }
```

5.2. Problema B- Buscando matriz

Dificultad: ⋆
Temas: Ad-hoc
Complejidad: $O(N)$

Entendiendo el problema

Dada una serie de N observaciones y a lo más S tipos de pago, se quiere encontrar una matriz de transiciones entre estos estados. Éstas son las probabilidades de cada estado por cada entrada de la matriz. Esta matriz no es simétrica.

Solución

Se recorre la cadena de observaciones, se identifica el tipo de observación o_i y o_{i-1}, y en una matriz se cuentan dichas incidencias, esta incidencia debe ser del tipo $s = o_{i-1} \rightarrow s' = o_i$. Entonces por cada una de estas incidencias podemos hacer `mat[s][s']++`. Al final cada renglón de la matriz se normaliza por el total de observaciones del tipo s.

Código

```
1   import java.util.*;
2
3   class Bmain
4   {
5           public static void main(String[] args)
6           {
7                   Scanner sc = new Scanner(System.in);
8                   int tt = sc.nextInt();
9
10                  while (tt-- > 0)
11                  {
12                          double mat[][] = new double[20][20], tot[] = new
    ↪   double[20];
13                          int arr[] = new int[505];
14                          int S, N;
15
16                          N = sc.nextInt();
17                          S = sc.nextInt();
```

```
18
19                          for(int j = 0; j < N; j++)
20                          {
21                                  arr[j] = sc.nextInt();
22                                  arr[j]--;
23                          }
24
25                          for(int j = 0; j < N-1; j++)
26                          {
27                                  tot[arr[j]]++;
28                                  mat[arr[j]][arr[j+1]]++;
29                          }
30
31                          for(int i = 0; i < S; i++)
32                          {
33                                  for(int j = 0; j < S; j++)
34                                  {
35                                          if(tot[i] > 0)
36                                                  System.out.printf("%.2f ",
    ↪  mat[i][j] / tot[i]);
37                                          else
38                                                  System.out.printf("%.2f ",
    ↪  mat[i][j]);
39                                  }
40                                  System.out.println();
41                          }
42                          System.out.println();
43                  }
44          }
45  }
```

5.3. Problema C - Contando nombres

Dificultad: ⋆⋆⋆
Temas: Matemáticas
Complejidad: $O(n^2)$

Entendiendo el problema

Dada una cadena de tamaño n, compuesta por letras minúsculas, se te pide saber cuántas cadenas diferentes puedes formar usando los caracteres de la cadena.

Solución

Sabemos que podemos acomodar los n caracteres de la cadena en $n!$ formas, creando este número de cadenas. Sin embargo existen algunas cadenas que se repiten, en especial, si a_i es la cantidad que se repite el i-ésimo tipo de caracter de la cadena, entonces se repiten $a_1!a_2!...a_k!$ cadenas. Por lo tanto la respuesta es $\frac{n!}{a_1!a_2!...a_k!} = \binom{n}{a_1}\binom{n-a_1}{a_2}...\binom{n-a_1-...-a_{k-1}}{a_k}$

Código

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

#define For(i, a, b) for(int i=a; i<b; ++i)
#define INF (1<<30)
#define MP make_pair
#define MOD 1000000007

using namespace std;

typedef pair <int, int> ii;
typedef long long ll;

int C[205][205], v[30];

int main()
{
    //std::ios_base::sync_with_stdio(false);

    For(i, 0, 201)
        For(j, 0, i+1)
            C[i][j] = i == j or j == 0 ? 1 : (C[i-1][j] + C[i-1][j-1]) % MOD;

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

    while (tt--)
    {
        memset(v, 0, sizeof v);

        char s[205];
        scanf("%s", s);

        int n = strlen(s);
        For(i, 0, n)
            v[s[i]-'a']++;

        int ans = 1;
        For(i, 0, 'z'-'a'+1)
        {
            if (v[i] > 0)
                ans = (ans*C[n][v[i]]) % MOD;
            n -= v[i];
        }

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

    return 0;
}
```

5.4. Problema D - Dados

Dificultad: ⋆⋆
Temas: Matemáticas
Complejidad: $O(1)$

Entendiendo el problema

Dados n dados, todos con m caras, quieres saber cuál es la suma de las caras de los dados con mayor probabilidad de salir al lanzar todos los dados.

Solución

Sea a y b el mínimo y máximo, respecticamente, valor posible para la suma. Entonces es fácil ver que la respuesta es $\frac{a+b}{2}$.

Código

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

#define For(i, a, b) for(int i=(a); i<(b); ++i)
#define INF 1000000000
#define MP make_pair

using namespace std;

typedef pair <int, int> ii;
typedef long long ll;

double p[55][2505];

int main()
{
    //std::ios_base::sync_with_stdio(false);

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

    while (tt--)
    {
        int n, m;
        scanf("%d %d", &n, &m);
        printf("%d\n", n == 1 ? 1 : (n*m - n)/2 + n);
    }

    return 0;
}
```

5.5. Problema E - Encriptando mensajes

Dificultad: ⋆
Temas: Implementación
Complejidad: $O(n)$

Entendiendo el problema

Dada una cadena s y un entero k, se te pide que encriptes s haciendo x XOR k para cada caracter x de s.

Solución

Directa.

Código

```
#include <cstdio>
#include <iostream>
#include <string>
#define For(x,a,b) for(int x=(a); x<(b); x++)
using namespace std;

int main()
{
    int C,k;
    string cad;

    scanf("%d",&C);

    For(c,0,C)
    {
        cin >> k;
        getline(cin,cad);
        getline(cin,cad);

        For(i,0,cad.size())
        {
            printf("%c ",(cad[i]^k));
        }
        printf("\n");
    }
    return 0;
}
```

5.6. Problema F - Focos

Dificultad: ⋆⋆
Temas: Matemáticas
Complejidad: $O(1)$

Entendiendo el problema

Se te da un arreglo de focos y un interruptor que, al accionarlo la i-ésima vez, cambia el estado de los focos que son múltiplos de i. Después de accionar n veces el interruptor, quieres saber el estado del n-ésimo foco, el cuál está inicialmente apagado.

Solución

Si el número de divisores de n es par, el foco estará apagado, sino estará prendido. Un entero n tiene un número par de divisores si y sólo si es un cuadrado perfecto.

Código

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

#define For(i, a, b) for(int i=a; i<b; ++i)
#define INF (1<<30)
#define MP make_pair

using namespace std;

typedef pair <int, int> ii;
typedef long long ll;

int main()
{
    //std::ios_base::sync_with_stdio(false);

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

    while (tt--)
    {
        int n;
        scanf("%d", &n);

        int r = sqrt(n);
        printf("%s\n", r*r == n ? "son unos genios" : "les hace falta estudiar");
    }

    return 0;
}
```

5.7. Problema G - Grupos

Dificultad: $\star$
Temas: Ad-hoc
Complejidad: $O(nm)$

Entendiendo el problema

Dados n grupos, donde deben haber m alumnos, nm alumnos y la personalidad a_i del i-ésimo alumno, quieres saber si es posible asignar los alumnos a los grupos de tal forma que no haya un grupo con más de un alumno de la misma personalidad.

Solución

Si existen $k > n$ alumnos con la misma personalidad, es imposible crear los grupos, en caso contrario es siempre posible.

Código

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

#define For(i, a, b) for(int i=a; i<b; ++i)
#define INF (1<<30)
#define MP make_pair

using namespace std;

typedef pair <int, int> ii;
typedef long long ll;

int a[100005];

int main()
{
    //std::ios_base::sync_with_stdio(false);

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

    while (tt--)
    {
        memset(a, 0, sizeof a);

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

        bool ok = true;
        For(i, 0, n*m)
        {
            int x;
            scanf("%d", &x);
            a[x]++;
            if (a[x] > n)
                ok = false;
        }

        printf("%s\n", ok ? "si" : "no");
    }

```

```
41      return 0;
42  }
```

Capítulo 6

Concurso de Programación de la XIV Semana de MAC

6.1. Problema A - Armando elecciones

Dificultad: ⋆⋆⋆
Temas: Teoría de Gráficas
Busquedas
Complejidad: $O(n + r)$

Entendiendo el problema

Para cada caso de prueba se debe construir una gráfica con las relaciones de amistad como aristas y como vértices las n personas. Cada candidato representa una componente de la gráfica.

Solución

Para cada candidato se realiza una búsqueda en amplitud y por cada vértice que puede alcanzar se suma la cantidad de puntos que ese vértice tiene. Se considera que el genio malo aporta $-\infty$ puntos y el bueno ∞. Al final se suman los puntos de aquellas personas que no pertecen a ninguna componente representada por algun candidato; si esta suma más los puntos del segundo candidato más alto (en puntos) es mayor o igual a los puntos del candidato más alto entonces no es posible determinar quién ganará las eleciones, en caso contrario las gana el candidato con más puntos.

Código

```
1 #include <iostream>
2 #include <queue>
3 #include <algorithm>
4 using namespace std;
```

```
5
6   queue <int> cola;
7   int matriz[505][505], votos[505], personas[505];
8   pair <int, int> candidatos[505];
9   int infinito=100000000;
10  int n;
11
12  int buscar()
13  {
14      int suma =0, a, i;
15      bool ganaste=false, perdiste=false;
16      while (!cola.empty() )
17      {
18          a=cola.front();
19          cola.pop();
20          if ( personas[a] ) continue;
21
22          if (a==0)   ganaste=true;
23          else if (a==n+1) perdiste=true;
24          else suma += votos[a];
25          personas[a]=1;
26          for (i=-1; ++i<=n+1; )
27              if (matriz[a][i]==1)
28                  cola.push(i);
29      }
30      if (ganaste && !perdiste) suma= infinito;
31      if (perdiste && !ganaste) suma =-infinito;
32      return suma;
33  }
34
35  int main()
36  {
37      int t, c, r, a, b, i, j;
38      cin>>t;
39      while (t--)
40      {
41          cin>>n>>c>>r;
42
43          //limpiar
44          while (!cola.empty() ) cola.pop();
45          for (i=-1; ++i<=n+1; )
46              for (j=-1; ++j<=n+1; )
47                  matriz[i][j]=0;
48          for (i=-1; ++i<=n+1; )
49              personas[i]=0;
50          for (i=-1; ++i<c; )
51              candidatos[i].first=0;
52          votos[0]=votos[n+1]=0;
53
54          for (i=0; ++i<=n; )
55              cin>>votos[i];
56
57          for (i=-1; ++i<c; )
58              cin>>candidatos[i].second;
59
60          while (r--)
61          {
```

```
62              cin>>a>>b;
63              if (a==-1) a=n+1;
64              if (b==-1) b=n+1;
65              matriz[a][b]=1;
66              matriz[b][a]=1;
67          }
68
69          for (i=-1; ++i<c; )
70          {
71              cola.push( candidatos[i].second );
72              candidatos[i].first = buscar();
73          }
74
75          sort(candidatos, candidatos+c);
76
77          if (candidatos[c-1].first==infinito  || candidatos[c-2].first==-infinito
    ↪  )
78              cout<<candidatos[c-1].second<<endl;
79          else
80          {
81              //contar los no amigos
82              int otrasuma=0;
83              for (i=0; ++i<=n; )
84                  if ( personas[i]==0 )
85                      otrasuma+=votos[i];
86              if ( (candidatos[c-2].first + otrasuma) >= candidatos[c-1].first )
87                  cout<<"Es mas facil salir de la friendzone que saber quien
    ↪  ganara\n";
88              else
89                  cout<<candidatos[c-1].second<<"'\n";
90          }
91      }
92      return 0;
93  }
```

6.2. Problema B - Buscando amigos

Dificultad: ⋆⋆⋆
Temas: Matemáticas
Teoría de Números
Complejidad: $O(n)$

Entendiendo el problema

Debemos saber si la suma de los divisores propios de un número a son iguales a un número b.

Solución

En este concuso se permitió la solucion "naive", la cual consisitia en verificar todos los números menores a a y sumar aquellos que lo dividen, después verificar si esa suma es igual a b. Sin embargo existe una mejor solucion:

Sean $p_1, p_2, \ldots, p_k$ los factores primos de a. Podemos decir que:

$$a = p_1^{f_1} \times p_2^{f_2} \times \ldots \times p_k^{f_k}$$

La suma de los divisores de a, incluyendo a a es:

$$S = \frac{p_1^{f_1+1} - 1}{p_1 - 1} \times \frac{p_2^{f_2+1} - 1}{p_2 - 1} \times \ldots \times \frac{p_k^{f_k+1} - 1}{p_k - 1}$$

Entonces basta con verificar si $S - a = b$. A pesar de que esta solución se ve mas compleja, computacionalmente es mejor y más rápida.

Código

```
#include <iostream>
using namespace std;
int factores[1000][2], i;

void descomponer(int a)
{
    int j =1;
    while (a!=1)
    {
        ++j;
        if (a%j==0)
        {
            ++i;
            factores[i][0]=j;
            while (a%j==0)
            {
                ++factores[i][1];
                a /=j ;
            }
        }
    }
}

int geometrica(int j)
{
    int k, res=1;
    for (k=-1; ++k<=factores[j][1]; )
        res *=factores[j][0];
    --res;
    res /=(factores[j][0]-1);
    return res;
}

int main()
{
    int t, a, b, j,m, res;
    cin>>t;
    while (t--)
    {
        cin>>a>>b;
        i=-1;
```

```
42
43          for (j=-1; ++j<1000; factores[j][1]=0);  //limpiar
44
45          descomponer(a);
46
47          res=1;
48          for (j=-1; ++j<=i;)
49          {
50              res *=geometrica(j);
51          }
52          res -=a;
53          if (res!=b)
54              cout<<"Llora desconsoladamente\n";
55          else cout<<"Quiza y sin quiza\n";
56      }
57  }
```

6.3. Problema C - Caballo Sin Mecate

Dificultad: ⋆⋆
Temas: Teoría de Grafos
Busquedas
Complejidad: $O(n+m)$

Entendiendo el problema

Debemos determinar el mínimo número de movimientos requeridos para que el caballo llegue a cierta pocisión del tablero, si es que ésto es posible.

Solución

El grafo es implícito y es el tablero de ajedrez, y decimos que un vértice (escaque) es vecino de otro si se puede llegar de uno a otro por medio de un movimiento de caballo, los vértices que tienen una pieza de ajedrez, que no es el caballo, en ellos no tienen vecinos. Para resolver el problema realizamos una búsqueda en amplitud desde la posición inicial del caballo contando los movimientos requeridos para llegar a cierto lugar, si en la búsqueda no se encuentra la posción deseada decimos que es imposible.

Código

```
1   import java.util.*;
2
3   class Cmain
4   {
5       static class pair
6       {
7           public int x, y;
8           public pair(int _x, int _y)
9           {
10              x = _x;
```

```
11              y = _y;
12          }
13      }
14
15      static int dX[] = {-2, -1, 1, 2, 2, 1, -1, -2}, dY[] = {1, 2, 2, 1, -1, -2,
    ↪  -2, -1};
16
17      public static void main(String[] args)
18      {
19          Scanner sc = new Scanner(System.in);
20          int tt = sc.nextInt();
21
22          while (tt-- > 0)
23          {
24              int n = sc.nextInt(), m = sc.nextInt();
25              String board[] = new String[n];
26
27              sc.nextLine();
28              for (int i = 0; i < n; ++i)
29                  board[i] = sc.nextLine();
30
31              int xf = sc.nextInt(), yf = sc.nextInt();
32              xf = n-xf;
33              --yf;
34
35              int dist[][] = new int[n][m];
36              Queue<pair> cola = new LinkedList<pair>();
37
38              for (int i = 0; i < n; ++i)
39                  for (int j = 0; j < m; ++j)
40                      if (board[i].charAt(j) == 'c')
41                      {
42                          cola.add(new pair(i, j));
43                          dist[i][j] = 1;
44                      }
45
46              while (!cola.isEmpty())
47              {
48                  pair act = cola.remove();
49
50                  if (act.x == xf && act.y == yf)
51                      break;
52
53                  for (int k = 0; k < 8; ++k)
54                  {
55                      int nx = act.x+dX[k], ny = act.y+dY[k];
56
57                      if (nx < 0 || nx >= n || ny < 0 || ny >= m ||
    ↪  board[nx].charAt(ny) != '*' || dist[nx][ny] != 0)
58                          continue;
59
60                      cola.add(new pair(nx, ny));
61                      dist[nx][ny] = dist[act.x][act.y] + 1;
62                  }
63              }
64
65              System.out.println(dist[xf][yf]-1);
```

```
66          }
67      }
68  }
```

6.4. Problema D - Doncella Jean

Dificultad: -⋆
Temas: Ad Hoc
Complejidad: $O(n)$

Entendiendo el problema

En este caso, la solución es indirecta, se debe de contar cuantas veces aparece la cadena ya sea "si" o "no" y determinar su proporción.

Solución

En el caso en el que la proporción de "no" sea ≥ 0.2 se imprime `friendzone`, en caso contrario `:)`.

Código

```
1   #include <iostream>
2   #include <cstdio>
3   #include <cstring>
4   #include <algorithm>
5   #include <cmath>
6   using namespace std;
7
8   #define For(x,a,b) for(unsigned int x = (a); x < (unsigned int)(b); x++)
9
10  typedef long long ll;
11
12
13  int main()
14  {
15          int C, n, x;
16          double r, tol = 0.2;
17          string cad;
18
19          scanf("%d",&C);
20
21          while(C--)
22          {
23                  scanf("%d", &n);
24                  x = 0;
25                  For(i,0,n){
26                          cin >> cad;
27                          x += cad == "no";
28                  }
29
30
31                  r = (double)x / (double)n;
```

```
32
33                  if(r < tol)
34                          printf(":)\n");
35                  else
36                          printf("friendzone\n");
37          }
38
39          return 0;
40  }
```

6.5. Problema E - Explanada

Dificultad: ⋆⋆⋆⋆
Temas: Matemáticas, Geometría
Complejidad: $O(n)$

Entendiendo el problema

Dado un polígono convexo de n vértices ($p_0, p_1, \ldots, p_{n-1}$ en el orden de las manecillas del reloj o inverso), debemos determinar si un punto a se encuentra dentro del polígono.

Solución

Este un problema bien conocido de geometría computacional que se puede resolver con el Counter Clockwise Test. Sean $\overrightarrow{ap_i}$ y $\overrightarrow{ap_{i+1}}$ dos vectores obtenidos de estos 3 puntos. EL producto cruz de $\overrightarrow{ap_i} \times \overrightarrow{ap_{i+1}}$ en otro vector que es perpendicular a ambos. Si la magnitud del vector resultante es positiva/cero/negativa, entonces tenemos que a se encuentra a la derecha/es colineal/izquierda, respectivamente. Entonces si se cumple que para todo i el vector resultante $\overrightarrow{ap_i} \times \overrightarrow{ap_{(i+1) \mod n}}$ tiene el mismo signo, quiere decir, que el punto siempre estuvo del mismo lado de las aristas y por lo tanto se encuentra dentro del polígono convexo.

Código

```
1   #include <bits/stdc++.h>
2   #define For(i, a, b) for(int i=(a); i<(b); ++i)
3   #define INF 1000000000
4   #define MP make_pair
5   #define x first
6   #define y second
7
8   using namespace std;
9
10  typedef long long ll;
11  typedef pair <int, int> ii;
12
13  ii p[105];
14  bool ccw(ii A, ii B, ii C)
```

```
15 {
16         return (B.x-A.x)*(C.y-A.y) - (B.y-A.y)*(C.x-A.x) >= 0;
17 }
18
19 int main()
20 {
21         int tt;
22         scanf("%d", &tt);
23         while (tt--)
24         {
25                 int n;
26                 scanf("%d", &n);
27                 For(i, 0, n)
28                         scanf("%d %d", &p[i].x, &p[i].y);
29                 ii f;
30                 scanf("%d %d", &f.x, &f.y);
31                 bool sgn = ccw(f, p[0], p[1]), ok = true;
32                 For(i, 1, n)
33                         if (ccw(f, p[i], p[(i+1)%n]) != sgn)
34                         {
35                                 ok = false;
36                                 break;
37                         }
38                 printf("%s\n", ok ? "si" : "no");
39         }
40
41         return 0;
42 }
```

6.6. Problema F - Fiesta

Dificultad: ⋆⋆⋆
Temas: Matemáticas
Probabilidad
Complejidad: $O(n * m)$

Entendiendo el problema

Debemos encontrar el valor esperado del número de personas que asistirán a la fiesta, el cual está dado por la siguiente ecuación:

$$E(X) = \sum_{x=0}^{n} x f(x)$$

donde $f(x)$ es la probabilidad de que asistan x personas a la fiesta.

Solución

Obtner $f(x)$ es algo complicado, ya que para poder obtener esto debemos encontrar todos los subconjuntos A de tamaño x del conjunto $[n] = \{1, 2, \ldots, n\}$. Después considerando a $f(x) = 1$, decimos que para cada subconjunto A, si

$i \in A$ entonces $f_A(x) = f_A(x) * p_i$ o en caso contrario $f_A(x) = f_A(x) * (1 - p_i)$. Y $f(x) = \sum f_A(x)$ Esta tarea puede sonar bastante complicada, aún así hemos probado que:

$$\begin{aligned} E(X) &= \sum_{i=1}^{n} p_i \\ &= \sum_{i=1}^{n} \sum_{j=1}^{m} \frac{r_{ij}}{m} \\ &= \frac{1}{m} \sum_{i=1}^{n} \sum_{j=1}^{m} \frac{1}{a_{ij}} \end{aligned}$$

Código

```cpp
#include <bits/stdc++.h>
using namespace std;
//look at my code my code is amazing
#define FOR(i, a, b) for (int i = int(a); i < int(b); i++)
#define FOREACH(it, a) for (typeof(a.begin()) it = (a).begin(); it != (a).end();
 ↪  it++)
#define ROF(i, a, b) for (int i = int(a); i >= int(b); i--)
#define REP(i, a) for (int i = 0; i < int(a); i++)
#define INF 1000000000
#define INFLL 1000000000000000000LL
#define ALL(x) x.begin(), x.end()
#define MP(a, b) make_pair((a), (b))
#define X first
#define Y second
#define EPS 1e-9
#define DEBUG(x)   cerr << #x << ": " << x << " "
#define DEBUGLN(x) cerr << #x << ": " << x << " \n"
typedef pair<int, int> ii;
typedef vector<int> vi;
typedef long long ll;
typedef vector<bool> vb;
//ios_base::sync_with_stdio(0);//fast entrada/salida ;-)

void solve()
{
    int n, m;
    scanf("%d %d", &n, &m);
    double ev = 0.0, p = 0.0;
    REP(i, n)
    {
        p = 0.0;
        REP(j, m)
        {
            int a;
            scanf("%d", &a);
            p += (a ? (1.0/a) : 0 );

        }
        ev += p/m;
```

```
39      }
40      printf("%.2lf\n", ev);
41  }
42
43  int main()
44  {
45      int T;
46      scanf("%d", &T);
47      REP(i, T)
48          solve();
49      return 0;
50  }
```

6.7. Problema G - Guíando a Edgar

Dificultad: ⋆⋆⋆
Temas: Teoría de Grafos **Complejidad:** $O(n^3)$

Entendiendo el problema

Dado un grafo con $1 \leq n \leq 50$ vértices, y una serie de consultas, donde cada consulta se compone por dos vértices s, t y una arista $\{u, v\}$. Se te pide encontrar la distancia más corta de s a t pasando por la arista $\{u, v\}$.

Solución

Sea $d(a, b)$ la distancia mínima entre los vértices a y b. La solución es

$$\text{mín}\left(d(s,u) + w(u,v) + d(v,t), d(s,v) + w(u,v) + d(u,t)\right)$$

donde $w(u, v)$ es el peso de la arista $\{u, v\}$. Como vamos a procesar varias consultas, es necesario obtener todas las distancias mínimas, lo que podemos hacer con Floyd Warshall.

Código

```
1   #include <bits/stdc++.h>
2
3   #define For(i, a, b) for(int i=(a); i<(b); ++i)
4   #define INF 1000000000
5   #define MP make_pair
6
7   using namespace std;
8
9   typedef long long ll;
10  typedef pair <int, int> ii;
11
12  int dist[55][55], AdjMat[55][55];
13
14  int main()
15  {
16      //std::ios_base::sync_with_stdio(false);
```

```
17
18      int tt;
19      scanf("%d", &tt);
20
21      while (tt--)
22      {
23          int n;
24          scanf("%d", &n);
25
26          For(i, 0, n)
27              For(j, 0, n)
28                  scanf("%d", &AdjMat[i][j]);
29
30          For(i, 0, n)
31              For(j, 0, n)
32                  dist[i][j] = AdjMat[i][j];
33
34          For(k, 0, n)
35              For(i, 0, n)
36                  For(j, 0, n)
37                      dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);
38
39          int Q;
40          scanf("%d", &Q);
41          while (Q--)
42          {
43              int a, b, c, d;
44              scanf("%d %d %d %d", &a, &b, &c, &d);
45              --a, --b, --c, --d;
46
47              printf("%d\n", min(dist[a][c]+AdjMat[c][d]+dist[d][b],
    ↪  dist[a][d]+AdjMat[d][c]+dist[c][b]));
48          }
49      }
50
51      return 0;
52  }
```

6.8. Problema H - Hurgando en el CEDETEC

Dificultad: ⋆⋆⋆
Temas: Matemáticas
Teoría de Números
Complejidad: $O(logn)$

Entendiendo el problema

Para poder determinar si Kenny podrá algún día llegar al cubículo del CEDETEC o no, tenemos que saber si la combinación lineal $ax + by = d$ tiene solciones enteras para todo número. Suena complicado, aun así veremos que es una sencilla ecuación diofántica.

Solución

La ecuación $ax + by = d$ tiene soluciones enteras *si y solo si* el $\text{mcd}(a, b)$ divide a d, y como d puede ser cualquier número, el único número que divide a todos es el 1, entonces hay solución si $\text{mcd}(a, b) = 1$, o en otras palabras si a y b son *primos relativos*. Utilizamos entonces el algoritmo de Euclides podemos obtener la solución.

Código

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

#define For(i, a, b) for(int i=(a); i<(b); ++i)
#define INF 1000000000
#define MP make_pair

using namespace std;

typedef long long ll;
typedef pair <int, int> ii;

int gcd(int a, int b){ return b ? gcd(b, a % b) : a; };

int main()
{
    //std::ios_base::sync_with_stdio(false);

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

    while (tt--)
    {
        int a, b;
        scanf("%d %d", &a, &b);
        printf("%s\n", gcd(a, b) == 1 ? "si" : "no");
    }

    return 0;
}
```