TC Asuncion 2013

Índice general

Capítulo 1

Algoritmos aceptados

1.1. Sandro's Book

Se empleó una estructura `unordered_map <string, int>` para almacenar todas las subcadenas posibles. Luego, navegando por la estructura, se comprobó cuál subcadena aparece la mayor cantidad de veces, y en caso de empate, se eligió la más larga.

1.2. Emoticons :-)

La estructura `Node` es un árbol de búsqueda, cuyas ramas representan a los caracteres de las palabras admitidas. Para admitir una palabra, se agregan las ramas necesarias, y se señala con una bandera el nodo correspondiente al último caracter de la palabra. Así, pueden buscarse todas las palabras (emoticones en este caso) a la vez en una sola iteración, sin importar el tamaño de ellas.

Al realizar la búsqueda, halla siempre el emoticón más largo a partir de la posición apuntada. Una vez hallado, busca a partir de la siguiente posición si hay un emoticón que es una subcadena del mismo. Si lo halla, repite el mismo procedimiento hasta que no pueda hallar una subcadena.

El final de la última subcadena hallada es el caracter que debe ser reemplazado por un espacio.

1.3. Pascal Library

Este problema fue fácil de resolver. El operador de bits `&` hizo todo el trabajo.

1.4. Bubble Maps

Este problema fue un grato desafío.

Lo primero fue asociar las letras p, q, r, s con los números 0; 1; 3; 2, respectivamente. Así, en lugar de observar el comportamiento de las coordenadas desde una perspectiva rotacional, se hace desde una perspectiva más cercana a la cartesiana. Además, se obtienen ciertas ventajas al trabajar con valores numéricos (que todo matemático disfruta explotar).

El siguiente paso fue descubrir el patrón de movimiento:

- Se comienza siempre con el nivel de zoom más cercano. Supóngase que se busca mover a la dirección d (arriba, abajo, izquierda o derecha).
- Si es factible sin salir del nivel de zoom, realiza el movimiento y obtiene la coordenada deseada. Sino, simula realizar el movimiento contrario (llámese $-d$ si se desea) e intenta realizar el movimiento d en el siguiente nivel de zoom (cuatro veces mayor que el último visitado).
- Si llega al último nivel de zoom, se realiza el mismo procedimiento si puede moverse a la dirección d. Sino, se deduce que la casilla a la que se quiere llegar está fuera del mapa.

Por último, se convierten las coordenadas numéricas nuevamente a las letras correspondientes, indicando con `<none>` las casillas que están fuera del mapa.

1.5. Degrees of Separation

Siguiendo el consejo dado en la clase, implementé el algoritmo de *Floyd-Warshall* para resolver el problema de las distancias. Como no podía haber más de 50 vértices en el grafo, la complejidad $O(V^3)$ ($50^3 = 125\,000$) no supuso un problema de rendimiento.

La parte que podría considerarse más complicada fue la de reconocer las repeticiones de los nombres. Este problema se pudo resolver con una estructura `map <string, int>`, la cual asociaba cada cadena con el índice correspondiente en la matriz del grafo.

1.6. Maximum Sum

Este problema tuvo bastantes errores graciosos (implementaciones que planteaban diferentes formas de resolver el problema, y sin embargo daban los mismos números incorrectos).

Se empleó programación dinámica para resolver el problema, guardando las sumas de las líneas horizontales (filas o subfilas, si pueden llamarse así).

Se diseñó también una función para obtener la suma de un rectángulo, la cual obtenía recursivamente la suma del rectángulo de altura inmediatamente inferior si el mismo no era una línea horizontal (altura 1). En cada llamada a la función, se comparaba el resultado con el máximo valor hallado previamente.

Así, llamando al mínimo de funciones necesarias para comprobar las sumas de todos los rectángulos, se obtenía la máxima suma. La programación dinámica

permitió ejecutar el algoritmo con complejidad $O(n^3)$ una vez se almacenaron las sumas de las líneas horizontales.

1.7. Closest Point Pair

Fue uno de los algoritmos que más tiempo tomó comprenderlos. No fue difícil comprender el paso de divide y conquistarás, mas sí costó bastantes dolores de cabeza y leer bastantes fuentes distintas comprender por qué era suficiente hace tan pocas comprobaciones al comparar dos puntos en distintas regiones.

Una vez comprendida la demostración (como dos semanas después), perdí un día porque recibía constantemente **Wrong Answer**. He buscado un problema similar en UVa, "The Closest Pair Problem", y no tuve muchos inconvenientes en lograr que aceptara mi solución. ¿Cuál era el problema? Que a pesar de haber leído muchas veces y sospechado, no había visto que los índices de la solución debían ser escritos en orden.

1.8. How Many?

Me enojé con este problema, porque buscaba una solución bien matemática, y al final recurrí a una poco elegante programación dinámica.

A diferencia de la solución propuesta en el material, no contaba si el último movimiento fue hacia arriba o hacia abajo, sino que, si necesitaba hacer el conteo de algún pico, desplazaba el punto hasta el lugar donde quedaría después de un movimiento hacia arriba y otro hacia abajo (dos lugares a la izquierda en mi algoritmo, ya que comienza desde la derecha).

Además, era necesario calcular cuándo llegaba a coordenadas desde las cuales sólo podía hacer un movimiento (para incrementar el contador), como la recta $y = x$, o puntos desde donde ya era imposible realizar un movimiento válido (retornando 0 para no incrementar el contador).

1.9. Diagonal

Otro problema matemático para divertirse. Sabiendo que un polígono de n lados tiene

$$d = \frac{n(n-1)}{2}$$

diagonales, bastaba deducir n en función a d en

$$d \leq \frac{n(n-1)}{2}.$$

Con un poco de álgebra, se llegaba a la desigualdad

$$n^2 - 3n - 2d \geq 0,$$

cuyas posibles soluciones son

$$n \geq \frac{3 + \sqrt{9 + 8d}}{2}$$

o

$$n \leq \frac{3 - \sqrt{9 + 8d}}{2}.$$

Como n y d son enteros positivos, se deduce que

$$\begin{aligned} \frac{3 - \sqrt{9 + 8d}}{2} &\leq \frac{3 - \sqrt{17}}{2} \\ &< \frac{3 - 4}{2} = -\frac{1}{2} \\ &< 0, \end{aligned}$$

por tanto la segunda solución no es una opción válida para n.

Como sólo queda la primera opción, simplemente basta calcular

$$n = f(d) = \left\lceil \frac{3 + \sqrt{9 + 8d}}{2} \right\rceil$$

para resolver el problema.

1.10. Lonesome Knight

Admito que mi solución fue poco elegante, quizás por la prisa por tener algo rápido de depurar.

Simplemente definí una matriz con los números de movimientos posibles en cada casilla, escritos a mano. Como desde muy pequeño practicaba bastante al ajedrez, no tuve mayores problemas en controlar que no hubieran equivocaciones.

1.11. Turn the Lights Off

Tomé mi código fuente de "Flip Game" y lo adapté de modo que hiciera lo mencionado en la clase: permutar todos los interruptores de la primera fila y comprobar si cada permutación permitía que se apagaran todas las luces empleando el resto de los interruptores.

1.12. Stars

Este problema resolví siguiendo los tips dados en clase, después de caer rendido miserablemente ante el intento de una solución en menos de 0.25 segundos.

Tomó su tiempo entender el algoritmo, algunas pruebas de escritorio para entender por qué funcionaba, pero una vez logrado eso, no supuso mucha dificultad implementarlo.

Con esta solución volví a enamorarme del *mergesort*.

1.13. Cyclic antimonotonic permutations

Seguí el consejo discutido en clase: aprovechar que a partir de dos ciclos se puede obtener un nuevo ciclo solamente alterando un enlace por ciclo (se separa en dos enlaces cada uno) y hacer la unión entre el ciclo actual de n elementos (los enteros del 1 al n) con el número $n+1$, de modo que se mantenga antimonótona (observando los elementos en posiciones $n-1$ y n); hasta incluir todos los números solicitados.

1.14. Ordering Tasks

Este fue el problema que resolví en la clase cuando todo el mundo resolvía "Lonesome Knight" (qué elección para el primer problema).

La idea fue implementar árboles donde cada nodo tenía dos conjuntos: el de padres y el de hijos (el hijo depende del padre para ejecutarse). Una vez dadas las dependencias, se buscaba el primer nodo sin padres, se quitaba a cada hijo suyo a éste en el conjunto de padres, y se eliminaba al nodo, para seguir la búsqueda con los demás (no sin antes imprimirlo), hasta que no hubiera más nodos (tareas pendientes).

1.15. What Goes Up

Tomé como referencia un algoritmo que permitía calcular la longitud de la *Longest Increasing Subsequence* (*LIS*), sin necesidad de saber cuál es.

A dicha implementación le di un pequeño retoque: en lugar de almacenar solamente los números, almacenaba un par cuyo primer elemento era el número a comparar y el segundo un puntero al último número que lo precede (que el algoritmo original me permitía saber implícitamente).

Una vez calculada la longitud de la *LIS*, bastaba navegar por los pares de números (comenzando por el último obtenido al calcular la longitud) para construir en tiempo lineal toda la lista sin necesidad de consumir mucha memoria.

1.16. The primary problem

Pensaba que era uno de los problemas más fáciles y me llevé una sorpresa mayúscula.

Ya había hecho en el pasado problemas de UVa que solicitaban trabajar con números primos e inclusive con la conjetura de Godlbach. Así que pensaba que podía tranquilamente adaptar un poco de cada código para tener el resultado deseado, y me encontré con un TLE de película.

Rápidamente comprobé que mi problema era la generación de primos. Me convenía tener dos tablas: una de primalidad (`true` si el elemento es primo, `false` si no lo es) y otra con la lista de números primos.

Ninguno de mis algoritmos me permitía con facilidad obtener ambas listas, y sin embargo me resistía a hacer la criba de Eratóstenes por considerarla poco eficiente.

Mucho después, resignado, decidí implementarla, con algunas optimizaciones, y comprobé que no sólo me daba servidas las dos tablas, sino que... ¡Era 500 veces más rápida que mis otros algoritmos!

De más está decir que pedí miles de disculpas a Eratóstenes por haber dudado de él.

Los códigos que pensaba me iban a funcionar fueron "543 - Goldbach's Conjecture.cpp" y "Prime Cuts" (el mismo que también correspondió trabajar).

1.17. Minimal coverage

Un hermoso algoritmo greedy.

Apenas al ingresar los datos descarté los segmentos que no podían formar parte de la solución, por estar fuera de rango. Después, los ordené en función a su extremo izquierdo (hice un caso de desempate, pero no fue realmente necesario).

Así como M era el supremo del intervalo, definí una variable m (inicialmente 0) que era el ínfimo. Así, navegaba desde el primer segmento hasta el último con extremo izquierdo no mayor que m, y guardaba el mayor extremo derecho que encontraba, cuyo valor final asignaría a m (dicho segmento formaría parte de mi lista).

Repetí este procedimiento con todos los segmentos aún no visitados hasta tener $m \geq M$. El número de iteraciones es el conteo la lista donde guardaba los segmentos extremos me permitió mostrar el resultado.

1.18. Prime Cuts

Este problema se resolvió en dos pasos.

El primero consistió en crear una lista de primos. Como se mencionó en "The primary problem", este algoritmo no resultó ser el más eficiente, aunque el problema no lo requería. Los números primos se generaron comprobando la primalidad de cada número mayor que ellos (aprovechando que ya tenía una lista de primos para ahorrar cálculos).

El siguiente paso fue calcular el índice de la mediana de los primeros N primos (cómo odié al que decidió que 1 también debía ser primo, aún me dan arcadas), y según su paridad deduje los índices de los extremos que necesitaba para listar los que pedía el enunciado.

1.19. Median on the Plane

Este problema lo habría hecho sin mirar los tips si leía con más atención y observaba que decía claramente que no habían tres puntos colineales.

Pero como seguí las indicaciones, lo único creativo que pude hacer es elegir el punto inferior al mismo tiempo que iba ingresando los puntos y hacer el algoritmo para obtener el ángulo teniendo en cuenta el punto de discontinuidad en la recta $x = c$ cuando dos puntos están a igual distancia del eje de las ordenadas.

1.20. Rabbit Hunt

Este problema se resolvió con bastante fuerza bruta.

Por cada par de puntos, se calculó la pendiente y se agregaron ambos puntos a una estructura `map <Point, set <Point> >`, donde el primer elemento correspondía a la pendiente (con un valor especial para evitar divisiones entre 0 si la recta que unía los puntos era vertical) y el segundo al conjunto de puntos que compartían esa pendiente.

Por último, bastaba contar cuál era el conjunto con más elementos para cerrar el problema.

1.21. Ecological Premium

No le falta mucho para competir con problemas como "A + B" como el más fácil.

El único truco del algoritmo se trataba de darse cuenta de que la variable `animals` no influía en el resultado de los cálculos.

1.22. Funny Game

Un algoritmo *minimax* fue todo lo necesario para resolver el problema.

Tomando como referencia los tips, mi algoritmo reslizó los siguientes pasos:

1. Si el aeropuerto no tenía conexiones disponibles, perdía.
2. Sino, explotaba el aeropuerto (anulaba las conexiones de los demás aeropuertos con éste) e intentaba viajar a cada aeropuerto.
3. Si desde uno de los aeropuertos se pierde el juego, podía deducir que tenía la estrategia ganadora.

Haciendo las búsquedas ordenadamente, también es fácil dar con el menor número de aeropuerto, tal como solicita el enunciado.

1.23. Flip Game

Como $2^{16} = 65\,536$ no es un valor de gran costo a nivel computacional, opté por hacer fuerza bruta con una máscara de bits (bastante similar a la opción 1 que se propuso en los tips).

El único detalle a resaltar fue que implementé un arreglo de 6×6 para evitar casos particulares al elegir una casilla en los bordes o las esquinas para efectuar un movimiento.

1.24. Longest path in a tree

Opté por usar el teorema que se mencionó en la clase.

Hice un recorrido DFS para calcular la profundidad máxima a partir de un nodo N, de modo que pudiera recordar al nodo M más alejado de N. Luego hice el DFS a partir de M y el número dado fue el resultado buscado.

1.25. Tornado!

Este problema no fue difícil, pero tenía sus trampas y había que estar atento.

Si entre dos postes de concreto había c postes rotos, se necesitaban $\left\lfloor \frac{c}{2} \right\rfloor$ postes de madera entre ellos. El total de postes se obtenía sumando todos los valores parciales al dar una vuelta en el recorrido.

La trampa estaba cuando no había postes de concreto. En ese caso, para n postes rotos, la solución es $\left\lfloor \frac{n-1}{2} + 1 \right\rfloor$.

1.26. Product of Digits

Otro algoritmo verdaderamente greedy.

Contaba primero cuántos factores 9 tenía N, y cada vez que hallaba un factor, lo dividía por ese número. Luego, procedía de la misma forma con los números $8; 7; \ldots; 2$ hasta que $N = 1$ o se me acabaran los dígitos.

Si se me acaban los dígitos y $N \neq 1$, entonces sabía que era imposible hallar el número solicitado. En cambio, si $N = 1$, imprimía cada dígito del 2 al 9, esta vez en orden creciente, la cantidad de veces que había hallado cada uno.

Dos casos que tomé de forma particular fueron el 0 y el 1, que inmediatamente devolvían 1 y 10, respectivamente.

1.27. Two Teams

Este problema parecía más difícil de buenas a primeras, pero facilitaba muchas cosas.

En efecto, el único caso que no tenía solución era cuando había personas sin amigos, puesto que:

- Si todo los amigos de una persona están en un equipo, elige el otro.
- Si una persona tiene amigos en ambos equipos, elige cualquiera con la certeza de que cumplirá las condiciones.

Así, el problema se convirtió en uno de coloreado de grafos, donde se buscan nodos sin pintar. Por cada nodo sin pintar, se le asigna un color, y a todos sus amigos sin pintar el otro color (y así, recursivamente, pudiendo eliminar las amistades para ganar tiempo).

Si hallabas una persona sin pintar y sin amigos, se deducía que era imposible formar los equipos. Y de poder formarse los equipos, contaba simplemente los nodos que fueron pintados de un color y los listaba.

1.28. Basic wall maze

Este problema resolví haciendo un recorrido BFS mediante una estructura `queue <Path>`, donde `Path` almacena la casilla donde se encuentra y toda la ruta recorrida hasta el momento (en una cadena).

Dado que había como mucho $6^2 = 36$ casillas por las cuales pasar, no cabía duda de que el algoritmo iba a terminar muy rápidamente.

La parte más "difícil" fue crear una estructura para el tablero que permitiera insertar las murallas cómodamente. En mi caso, elegí hacer una matriz grande, donde dos casillas adyacentes tienen, en la matriz implementada, una casilla entre ellas, que representa la existencia de la muralla. Luego de hacer la conversió de coordenadas correspondiente, el problema estuvo resuelto.

1.29. Rope

El problema consistió en sumar las distancias de los segmentos y los arcos que se formaban en la figura.

Cada segmento es tangente a dos circunferencias, y fácilmente se demuestra que mide lo mismo que la distancia entre los centros de dichas circunferencias.

Para sumar los arcos, primero intenté calcular los ángulos que se formaban en cada arco, pero me salía tan desprolijo que decidí consultar los tips. ¿Cómo no se me había ocurrido que todos los arcos suman un giro? Podía haberlo deducido si observaba que cada uno tenía la amplitud de un ángulo externo del polígono delimitado por los centros.

1.30. Binary Lexicographic Sequence

En este problema disfruté crear un artificio matemático que me permitió dar con cada dígito en el orden que necesitaba imprimir. Sí, complejidad $O(n)$, donde n es el número de dígitos a mostrar.

Me interesaba definir una función $f(n)$ que indicara todos los números que podía escribir, siguiendo las reglas, sin tener un 1 en una posición mayor que n, contando de derecha a izquierda, donde la posición más a la derecha es 1 y no

0. No me tomó mucho tiempo deducir que la función debía ser:

$$f(n) = \begin{cases} 1 & \text{si } n = 0 \\ 2 & \text{si } n = 1 \\ f(n-1) + f(n-2) & \text{si } n \geq 2 \end{cases}$$

Cuando $n \geq 2$, $f(n-1)$ representa todos los números anteriormente contados y $f(n-2)$ son todos los números a los cuales se puede agregar un 1 en la posición n para formar los nuevos números sin tener dos dígitos 1 consecutivos.

Una vez definida la función, con programación dinámica se delimitaron los intervalos donde era seguro que el 1 de mayor valencia estaba en la posición n. Así, era fácil comprobar si el número ingresado estaba dentro del rango: si el número k era mayor que $f(k)$, se devolvía -1 y terminaba la ejecución. Sino, se hacían los siguientes pasos mientras la cantidad de dígitos n fuera positiva:

1. Si $k \leq f(n-1)$, imprimir un 0.
2. Sino, imprimir un 1 y a k restar $f(n-1)$.
3. Restar una unidad a n después de efectuar cualquiera de los casos.

En el momento que $n = 0$, se sabe que ya se imprimieron todos los números.

1.31. Distinct Subsequences

Este fue otro problema muy bonito de programación dinámica que disfruté matemáticamente hacerlo.

El truco consistió en numerar las posiciones de las letras y, para cada letra, guardar la siguiente información:

- Las subsecuencias formadas antes de incluir a la letra en la cadena (de ahora en adelante, `previous`).
- Las subsecuencias formadas incluyendo a la letra de la cadena (`count`).

Al iniciar el algoritmo, el atributo `count` para cada letra es 0. El mismo valor tomará inicialmente `previous`. Es fácil deducir que el valor de `count` para la letra en la iteración $n-1$ será el valor de `previous` para la letra en la posición n. Además, en caso de que una letra aparezca repetidas veces, sólo nos importan los valores de su última aparición.

Para calcular el valor de `count` para la letra c en la posición n, se realizan los siguientes pasos:

1. Se asigna a `count` el doble del valor de `count` para la letra en la posición $n-1$ (cada permutación anterior vuelve a contarse, esta vez incluyendo a la nueva letra en el extremo).

2. Se resta el número de subsecuencias que se formaron antes de la última aparición de c anterior a la actual, es decir, el último `previous` guardado para c (pues son estos los casos en los cuales no podría distinguirse cuál de las dos c se posicionó en el extremo).

3. Terminado de calcular `count`, se calcula el nuevo valor de `previous` como se dijo anteriormente.

Terminado el procedimiento para la última letra, su atributo `count` tiene el valor que estábamos buscando. Lo más interesante fue descubrir que sólo necesitaba tener tanto espacio de memoria como letras posibles a escribirse, pues permite hacer un algoritmo de complejidad $O(n)$ usando un espacio de memoria de tamaño constante.

Observaciones:

- En el algoritmo, cada operación numérica fue adaptada de tal modo que cumpla con el requisito de que el resultado final sea un valor en el intervalo $[0; 1\,000\,000\,007)$.
- En la línea 31 del código fuente (página 100), puede apreciarse la condición **if** (sub[c].count). Al revisar el código, creo que no es en realidad necesario considerar esta condición, pero lo conservo como tal porque fue así como lo envié al juez virtual.

1.32. Edit distance

Para este problema, no tuve tanta facilidad al plantearlo, y no me quedó otra que aprender el algoritmo de *Levenshtein* e implementarlo una vez entendido.

Personalmente, aún me quedan dudas en la demostración de por qué el artificio de agregar espacios entre algunas letras es clave a la hora de resover el problema. Es algo que me queda por investigar.

1.33. Cutting Sticks

Este problema parecía bastante atacable con algún artificio matemático, pero más tarde que temprano me vi obligado a tomar el camino que menos quería: probar todas las permutaciones posibles (aunque la programación dinámica lo hiciera más eficiente).

En un primer intento, quería trabajar con los trozos que debían formarse (si tengo una barra de 10 metros y la corto en los lugares 2; 4 y 7 tenía trozos de 2; 2; 3 y 3 metros, en ese orden). Sin embargo, la programación dinámica era costosa porque necesitaba almacenar un vector para cada permutación de trozos, lo cual tampoco era muy eficiente.

Una vez descartado el caso anterior, recurrí al consejo dado en el PDF entregado en clase, y di con el resultado.

Una vez más aprendí que no debo mostrarme tan reacio a tomar el camino aparentemente ingenuo.

1.34. Flowers Flourish from France

Este problema no fue difícil de resolver.

Había que hallar la priemra letra de cada palabra, convertirla a mayúsculas y controlar si siempre se repetía la misma letra hasta llegar a un salto de línea.

La parte más tediosa del algoritmo fue hacer el parseo controlando que no hubiera errores de desatención.

Capítulo 2

Algoritmos finalizados pero no aceptados

2.1. Maximum Square

Este problema no pudo ser enviado porque el juez indicó constantemente **Submission Error** (es horrible cuando pasa eso).

Mi primer algoritmo era una versión bastante costosa, sin programación dinámica, que se basaba en hacer una lista de todas las casillas con valor 1 y, para cada casilla, comprobar el mayor cuadrado que se podía formar asumiendo que dicha casilla era la esquina superior izquierda. Previamente calculaba el mayor número posible del lado (el mínimo entre la raíz cuadrada de la cantidad de unos, la cantidad de filas y la cantidad de columnas de la matriz) para poder detener la búsqueda en caso de hallarla.

Busqué los test cases oficiales y descubrí que era poco eficiente el algoritmo (le tomó más de 18 minutos terminar el recorrido), y en ese momento me animé a mirar los consejos en el PDF de la clase. Implementé el algoritmo recomendado y, al probarlo comprobé que efectivamente era mucho más veloz, pero algunas impresiciones me llevaban a no hallar el mayor de los cuadrados (más tarde comprobé que era porque necesitaba hacer la recursión aún si mi casilla era un 0). Corregido el error, el algoritmo funcionó tal como esperaba.

Lastimosamente, me quedé con las ganas de saber si realmente era correcta mi solución.

2.2. Turkish Roulette

Este problema me dejó pensando bastante tiempo. Lo primero que pensé fue en optimizar el uso de las casillas precalculando las sumas.

Luego me puse a pensar de qué modo podía optimizar el cálculo del mejor caso. Después de bastantes planteamientos, no hallé uno que me convencía, y

recurrí a la recomendación en el PDF.

Particularmente no veía cómo podía haber mucha optimización con esa técnica, ya que depende de tres variables: el número de bolas que ubicar, la posición disponible y las posiciones que permite la primera bola. Esta última no logré representar de una muy buena forma en programación dinámica, y me quedó un algoritmo de complejidad $O(n^3)$, ya que necesitaba borrar los resultados guardados en cada test case.

Sin embargo, después de un intento que resultó en **Time Limit Exceeded**, logré hacer un pequeño cambio que me ayudó a ganar tiempo (alrededor de 2.5 segundos), pero la respuesta fue **Wrong Answer**.

Revisé dónde podía haber fallado mi implementación, pero no pude dar con la falla durante la competencia.

Capítulo 3

Algoritmos incompletos

3.1. Onion Layers

Leí el algoritmo de búsqueda de *Graham*, el cual no sólo me pareció interesante por su complejidad ($O(n \log n)$ al ordenar y $O(n)$ al elegir el conjunto de puntos), sino que pensaba que podía adaptar mi solución de "Median on the Plane" para llegar a la solución.

Pero tenía sólo 40 minutos para que acabara la competencia, y no me alcanzó el tiempo diseñar la parte correspondiente a la eliminación de los puntos ya hallados en el convex hull. Por tanto, quedó incompleto el algoritmo.

Apéndice A

Algoritmos aceptados

A.1. Sandro's Book

```
1  #include <iostream>
2  #include <string>
3  #include <unordered_map>
4
5  using namespace std;
6
7  typedef unordered_map <string, int> Map;
8  typedef Map::iterator Iterator;
9
10 Map used_substr;
11
12
13 int main ()
14 {
15     string st, sub;
16     int s, l, i, j;
17     int max_used;
18
19     cin >> st;
20
21     s = st.size();
22     for (i = 0; i < s; ++i)
23     {
24         l = s - i;
25         for (j = 1; j <= l; ++j)
26         {
27             sub = st.substr (i, j);
28             used_substr[sub]++;
29         }
30     }
31
32     max_used = 0;
33     sub = "";
34     for (Iterator it = used_substr.begin(); it !=
           used_substr.end(); ++it)
35     {
36         if (it->second > max_used)
37         {
38             max_used = it->second;
39             sub = it->first;
40         }
41         else if (it->second == max_used && it->first.size
               () > sub.size())
```

```
42          {
43              sub = it->first;
44          }
45      }
46
47      cout << sub << endl;
48
49      return 0;
50  }
```

A.2. Emoticons :-)

```
1  #include <iostream>
2  #include <map>
3  #include <string>
4
5  using namespace std;
6
7  struct Node;
8
9  typedef map <char, Node *> Map;
10 typedef Map::iterator Iterator;
11
12
13 struct Node
14 {
15     bool emoticon;
16     Map next;
17
18     Node () : emoticon(false) {}
19
20     ~Node () {
21         for (Iterator it = next.begin(); it != next.end()
               ; ++it)
22         {
23             delete it->second;
24         }
25     }
26
27
28     Node * next_node (char c)
29     {
30         Iterator it = next.find (c);
31         return it == next.end() ? 0 : it->second;
32     }
33
34
35     Node * insert (char c)
36     {
37         Node *n = new Node;
38         next[c] = n;
39         return n;
40     }
41
42     Node * go_next (char c)
```

```
43      {
44          Node *n;
45          Iterator it = next.find (c);
46          if (it == next.end())
47          {
48              n = insert (c);
49              next[c] = n;
50              return n;
51          }
52          else
53          {
54              return it->second;
55          }
56      }
57
58
59      Node * insert (const char *s)
60      {
61          Node *n = this;
62          for (; *s; ++s)
63          {
64              n = n->go_next (*s);
65          }
66          n->emoticon = true;
67          return n;
68      }
69
70      int find_emoticon (const char *s)
71      {
72          Node *n = this;
73          int r = 0, c = 0;
74          for (; *s; ++s)
75          {
76              n = n->next_node (*s);
77              if (n == 0)
78              {
79                  break;
80              }
81              ++c;
82              if (n->emoticon)
83              {
84                  r = c;
85                  c = 0;
86                  break;
87              }
88          }
```

```
89          if (r)
90          {
91              return r;
92          }
93          return r;
94      }
95
96      int count_emoticons (char *s)
97      {
98          int i, r = 0, m, f;
99          while (*s)
100         {
101             m = find_emoticon (s);
102             if (m)
103             {
104                 for (i = 1; i < m; ++i)
105                 {
106                     f = find_emoticon (s + i);
107                     if (f)
108                     {
109                         m = min (m, i + f);
110                         if (f == 1)
111                             break;
112                     }
113                 }
114                 ++r;
115                 s += m;
116             }
117             else
118                 ++s;
119         }
120         return r;
121     }
122 };
123
124
125 int main ()
126 {
127     int n, m, count;
128     string line;
129
130     while (cin >> n >> m, n || m)
131     {
132         Node emoticon;
133         while (n--)
134         {
```

```
135             cin >> line;
136             emoticon.insert (line.c_str());
137         }
138         count = 0;
139         while (cin.get() != '\n');
140         while (m--)
141         {
142             getline(cin,line,'\n');
143             count += emoticon.count_emoticons (&line[0]);
144         }
145         cout << count << endl;
146     }
147 
148     return 0;
149 }
```

A.3. Pascal Library

```
1  #include <iostream>
2  #include <algorithm>
3
4  using namespace std;
5
6  #define MAX_N 100
7  #define MAX_D 500
8
9  int alumni[MAX_N];
10
11
12 inline bool someone_participated_always (int N)
13 {
14     for (int n = 0; n < N; ++n)
15     {
16         if (alumni[n])
17             return true;
18     }
19     return false;
20 }
21
22
23 int main ()
24 {
25     int N, D, n, d, v;
26
27     while (cin >> N >> D, N || D)
28     {
29         fill (alumni, alumni + N, 1);
30         for (d = 0; d < D; ++d)
31         {
32             for (n = 0; n < N; ++n)
33             {
34                 cin >> v;
35                 alumni[n] &= v;
36             }
37         }
38         cout << (someone_participated_always (N) ? "yes\n
               " : "no\n");
39     }
40
41     return 0;
42 }
```

A.4. Bubble Maps

```
1  #include <iostream>
2  #include <cctype>
3  using namespace std;
4
5  #define MAX_LENGTH 5001
6  #define END -1
7
8
9  int to_number[256];
10 char to_char[4] = {'p', 'q', 's', 'r'};
11
12 struct Region
13 {
14
15     int c[MAX_LENGTH];
16     int length;
17
18     inline int & operator [] (unsigned int i) {return c[i
           ];}
19     inline int operator [] (unsigned int i) const {return
           c[i];}
20
21     inline void operator = (const Region &r)
22     {
23         length = r.length;
24         for( int i = 0; i < length; ++i )
25         {
26             c[i] = r[i];
27         }
28     }
29
30     inline void next_up ( Region &r )
31     {
32         int i;
33         r = *this;
34         for( i = length - 1; i; --i )
35         {
36             if( r[i] > 1 )
37             {
38                 r[i] -= 2;
39                 break;
40             }
41             else
```

```
42              {
43                  r[i] += 2;
44              }
45          }
46          if( i == 0 )
47          {
48              if( r[i] > 1 )
49              {
50                  r[i] -= 2;
51              }
52              else
53              {
54                  r[i] = END;
55              }
56          }
57      }
58
59      inline void next_down ( Region &r )
60      {
61          int i;
62          r = *this;
63          for( i = length - 1; i; --i )
64          {
65              if( r[i] < 2 )
66              {
67                  r[i] += 2;
68                  break;
69              }
70              else
71              {
72                  r[i] -= 2;
73              }
74          }
75          if( i == 0 )
76          {
77              if( r[i] < 2 )
78              {
79                  r[i] += 2;
80              }
81              else
82              {
83                  r[i] = END;
84              }
85          }
86      }
87
```

```
 88      inline void next_left ( Region &r )
 89      {
 90          int i;
 91          r = *this;
 92          for( i = length - 1; i; --i )
 93          {
 94              if( r[i] %2 == 1 )
 95              {
 96                  --r[i];
 97                  break;
 98              }
 99              else
100              {
101                  ++r[i];
102              }
103          }
104          if( i == 0 )
105          {
106              if( r[i] %2 == 1 )
107              {
108                  --r[i];
109              }
110              else
111              {
112                  r[i] = END;
113              }
114          }
115      }
116
117      inline void next_right ( Region &r )
118      {
119          int i;
120          r = *this;
121          for( i = length - 1; i; --i )
122          {
123              if( r[i] %2 == 0 )
124              {
125                  ++r[i];
126                  break;
127              }
128              else
129              {
130                  --r[i];
131              }
132          }
133          if( i == 0 )
```

```
134             {
135                 if( r[i] %2 == 0 )
136                 {
137                     ++r[i];
138                 }
139                 else
140                 {
141                     r[i] = END;
142                 }
143             }
144         }
145     };
146
147     istream & operator >> ( istream &is, Region &r )
148     {
149         char c;
150         do
151         {
152             c = is.get();
153         }
154         while( c != 'm' );
155         r.length = 0;
156         while( c = is.get(), !isspace( c ) )
157         {
158             r[r.length] = to_number[c];
159             ++r.length;
160         }
161         return is;
162     }
163
164     ostream & operator << ( ostream &os, const Region &r )
165     {
166         int i;
167         if( r[0] == END )
168             os << "<none>";
169         else
170         {
171             os << 'm';
172             i = 0;
173             do
174             {
175                 os << to_char[ r[i] ];
176                 ++i;
177             }
178             while( i < r.length );
179         }
```

```
180     return os;
181 }
182
183
184 inline void init ()
185 {
186     to_number['p'] = 0;
187     to_number['q'] = 1;
188     to_number['r'] = 3;
189     to_number['s'] = 2;
190 }
191
192
193 Region region, neighbor;
194
195
196 int main ()
197 {
198     int test_cases;
199
200     init();
201
202     cin >> test_cases;
203     while( test_cases-- )
204     {
205         cin >> region;
206         region.next_up( neighbor );
207         cout << neighbor;
208         region.next_down( neighbor );
209         cout << ' ' << neighbor;
210         region.next_left( neighbor );
211         cout << ' ' << neighbor;
212         region.next_right( neighbor );
213         cout << ' ' << neighbor;
214         cout << '\n';
215     }
216
217     return 0;
218 }
```

A.5. Degrees of Separation

```
1  #include <iostream>
2  #include <string>
3  #include <map>
4  using namespace std;
5
6  #define MAX_P 50
7  #define INF 1000000
8  typedef map <string, int> Map;
9  typedef Map::iterator Iterator;
10
11 int graph[MAX_P][MAX_P], P;
12
13
14 inline void floyd ()
15 {
16     int path;
17     for (int k = 0; k < P; ++k)
18     {
19         for (int i = 0; i < P; ++i)
20         {
21             for (int j = 0; j < P; ++j)
22             {
23                 int &gij = graph[i][j];
24                 path = graph[i][k] + graph[k][j];
25                 if( gij > path )
26                     gij = path;
27             }
28         }
29     }
30 }
31
32
33
34
35
36 struct Index
37 {
38     Map map;
39     size_t s;
40
41     Index() : s(0) {}
42
43     inline void clear () {
```

```
44          s = 0;
45          map.clear();
46      }
47
48      inline int & operator [] ( const string &str ) {
49          pair <Iterator, bool> p;
50          Iterator &it = p.first;
51          it = map.find( str );
52          if( it == map.end() ) {
53              pair <string, int> ins( str, s++ );
54              p = map.insert( ins );
55          }
56          return it->second;
57      }
58
59      inline int & index( const string &str ) {
60          return operator [] ( str );
61      }
62
63      inline void add_relationship( const string &str_i,
            const string &str_j ) {
64          size_t i, j;
65          i = index( str_i );
66          j = index( str_j );
67
68          graph[i][j] = graph[j][i] = 1;
69      }
70  };
71
72
73  Index index;
74
75
76  inline void reset ()
77  {
78      index.clear();
79      for( int i = 0; i < P; ++i )
80      {
81          for( int j = 0; j < i; ++j )
82          {
83              graph[i][j] = INF;
84          }
85          for( int j = i + 1; j < P; ++j )
86          {
87              graph[i][j] = INF;
88          }
```

```
89      }
90  }
91
92
93  int main ()
94  {
95      int R, gij, network;
96      string str_i, str_j;
97
98      network = 0;
99      while( cin >> P >> R, P || R )
100     {
101         ++network;
102         reset();
103         while( R-- )
104         {
105             cin >> str_i >> str_j;
106             index.add_relationship( str_i, str_j );
107         }
108         floyd();
109         R = 0;
110         for (int i = 0; i < P; ++i)
111         {
112             for (int j = 0; j < P; ++j)
113             {
114                 gij = graph[i][j];
115                 if( R < gij)
116                     R = gij;
117             }
118         }
119         cout << "Network " << network << ": ";
120         if( R >= INF )
121             cout << "DISCONNECTED\n\n";
122         else
123             cout << R << "\n\n";
124     }
125
126     return 0;
127 }
```

A.6. Maximum Sum

```
1  #include <iostream>
2  using namespace std;
3
4
5  #define MAX_N 100
6  #define MIN_VALUE (-256)
7  #define MIN_SUM (MAX_N * MAX_N * MIN_VALUE)
8
9
10 struct Sum
11 {
12     int value;
13     bool updated;
14 };
15
16
17 int maximum_sum = MIN_SUM;
18
19
20 int array[MAX_N][MAX_N];
21 Sum store_row_sum[MAX_N][MAX_N][MAX_N] = {0};
22
23
24 inline void compare_maximum_sum (int s)
25 {
26     if (maximum_sum < s)
27         maximum_sum = s;
28 }
29
30
31 int row_sum (int x, int y, int l)
32 {
33     Sum &s = store_row_sum[x][y][l];
34
35     if (s.updated)
36         return s.value;
37
38     s.value = l > 1
39         ? row_sum (x, y, l - 1)
40         : 0;
41     s.value += array[x][y + l - 1];
42
43     compare_maximum_sum (s.value);
```

```
44
45      s.updated = true;
46
47      return s.value;
48  }
49
50
51  int rect_sum (int x, int y, int l, int a)
52  {
53      int s;
54      s = a > 1
55          ? rect_sum (x + 1, y, l, a - 1)
56          : 0;
57      s += row_sum (x, y, l);
58
59      compare_maximum_sum (s);
60
61
62
63      return s;
64  }
65
66
67  int main ()
68  {
69      int n;
70
71      cin >> n;
72      for (int i = 0; i < n; ++i)
73      {
74          for (int j = 0; j < n; ++j)
75          {
76              cin >> array[i][j];
77          }
78      }
79
80
81
82      for (int y = 0; y < n; ++y)
83      {
84          for (int l = 1; l <= n - y; ++l)
85          {
86              for (int a = 1; a <= n; ++a)
87              {
88                  rect_sum (0, y, l, a);
89
```

```
90
91                  }
92              }
93          }
94
95          cout << maximum_sum << endl;
96
97          return 0;
98      }
```

A.7. Closest Point Pair

```
1  #include <iostream>
2  #include <iomanip>
3  #include <algorithm>
4  #include <cmath>
5  using namespace std;
6
7  #define MAX_N 50000
8  #define MAX_D 10000000000000
9
10 typedef long long int dist;
11 typedef dist coord;
12
13
14 struct Punto
15 {
16     coord x, y;
17     size_t ref;
18 };
19
20
21 bool min_x (const Punto &A, const Punto &B)
22 {
23     return A.x < B.x || (A.x == B.x && A.y < B.y);
24 }
25
26
27 bool min_y (const Punto &A, const Punto &B)
28 {
29     return A.y < B.y || (A.y == B.y && A.x < B.x);
30 }
31
32
33 bool operator < (const Punto &A, const Punto &B)
34 {
35     return min_x( A, B );
36 }
37
38
39 dist distancia (const Punto &A, const Punto &B)
40 {
41     dist x, y;
42     x = A.x - B.x; x *= x;
43     y = A.y - B.y; y *= y;
```

```
44      return x + y;
45  }
46
47
48  istream & operator >> (istream &is, Punto &P)
49  {
50      return is >> P.x >> P.y;
51  }
52
53
54  ostream & operator << (ostream &os, const Punto &p)
55  {
56      return os << '(' << p.x << ",␣" << p.y << ')';
57  }
58
59
60  struct Par
61  {
62      dist d;
63      size_t a, b;
64  };
65
66
67  bool operator < (const Par &p1, const Par &p2)
68  {
69      return p1.d < p2.d;
70  }
71
72
73  ostream & operator << (ostream &os, const Par &p)
74  {
75      if( p.a > p.b )
76          os << p.b << '␣' << p.a;
77      else
78          os << p.a << '␣' << p.b;
79      return os << '␣' << sqrt( (double) p.d );
80  }
81
82
83  inline void menor_distancia_bruta (const size_t N, const
        Punto P[], Par &r)
84  {
85      switch( N )
86      {
87          case 2:
88          {
```

```
89              r.a = P[0].ref;
90              r.b = P[1].ref;
91              r.d = distancia( P[0], P[1] );
92          }
93          break;
94
95          case 3:
96          {
97              dist d;
98              menor_distancia_bruta (2, P, r );
99              d = distancia( P[0], P[2] );
100             if( d < r.d )
101             {
102                 r.d = d;
103                 r.b = P[2].ref;
104             }
105             d = distancia( P[1], P[2] );
106             if( d < r.d )
107             {
108                 r.d = d;
109                 r.a = P[1].ref;
110                 r.b = P[2].ref;
111             }
112         }
113         break;
114
115         default:
116         {
117             const Punto
118                 * const pN = P + N,
119                 * const pn = pN - 1;
120             const Punto *pi, *pj;
121             dist d;
122             menor_distancia_bruta( 2, P, r );
123             for( pi = P; pi < pn; ++pi )
124             {
125                 for( pj = pi + 1; pj < pN; ++pj )
126                 {
127                     d = distancia( *pi, *pj );
128                     if( d < r.d )
129                     {
130                         r.d = d;
131                         r.a = pi->ref;
132                         r.b = pj->ref;
133                     }
134                 }
```

```
135             }
136         }
137     }
138 }
139
140
141 void menor_distancia (const size_t N, const Punto X[],
        const Punto Y[], Par &r)
142 {
143     if( N < 4 )
144     {
145         menor_distancia_bruta( N, X, r );
146     }
147     else
148     {
149         const size_t
150             Nizq = (N + 1) / 2,
151             Nder = N - Nizq;
152         size_t iz, de, vN, vn, lim;
153         dist d;
154         Punto X_izq[Nizq], Y_izq[Nizq], X_der[Nder],
                Y_der[Nder], V[N],
155             mediana;
156         Par r_izq, r_der;
157
158         for( size_t i = 0; i < Nizq; ++i )
159             X_izq[i] = X[i];
160         mediana = X[Nizq - 1];
161         for( size_t i = 0; i < Nder; ++i )
162             X_der[i] = X[Nizq + i];
163         iz = de = 0;
164         for( size_t i = 0; i < N; ++i )
165         {
166             if( mediana < Y[i] )
167                 Y_der[de++] = Y[i];
168             else
169                 Y_izq[iz++] = Y[i];
170         }
171
172         menor_distancia( Nizq, X_izq, Y_izq, r_izq );
173         menor_distancia( Nder, X_der, Y_der, r_der );
174         r = min( r_izq, r_der );
175
176         vN = 0;
177         for( size_t i = 0; i < N; ++i )
178         {
```

```
179             if( abs( Y[i].x - mediana.x ) < r.d )
180             {
181                 V[vN++] = Y[i];
182             }
183         }
184         if( vN > 1 )
185         {
186             vn = vN - 1;
187             for( size_t i = 0; i < vn; ++i )
188             {
189                 lim = min( i + 8, vN );
190                 for( size_t j = i + 1; j < lim; ++j )
191                 {
192                     d = distancia( V[i], V[j] );
193                     if( d < r.d )
194                     {
195                         r.d = d;
196                         r.a = V[i].ref;
197                         r.b = V[j].ref;
198                     }
199                 }
200             }
201         }
202     }
203 }
204
205
206 Punto X[MAX_N], Y[MAX_N];
207
208
209 int main ()
210 {
211     size_t N, i;
212     Par p;
213
214     cin >> N;
215     for( i = 0; i < N; ++i )
216     {
217         cin >> X[i];
218         X[i].ref = i;
219         Y[i] = X[i];
220     }
221     sort( X, X + N, min_x);
222     sort( Y, Y + N, min_y);
223     menor_distancia( N, X, Y, p );
224     cout << fixed << setprecision( 6 ) << p << endl;
```

```
225
226         return 0;
227     }
```

A.8. How Many?

```
1  #include <iostream>
2
3
4  using namespace std;
5
6  #define MAX_LENGTH 40
7  #define MAX_HEIGHT 20
8  #define MAX_PEAKS 20
9  #define MAX_HEIGHT_PEAK 20
10
11 typedef long long int uint;
12
13
14
15
16
17 uint paths (int x, int y, int r, int h)
18 {
19     static uint path[MAX_LENGTH][MAX_HEIGHT][MAX_PEAKS][
           MAX_HEIGHT_PEAK] = {0};
20     static bool updated[MAX_LENGTH][MAX_HEIGHT][MAX_PEAKS
           ][MAX_HEIGHT_PEAK] = {false};
21
22
23
24     uint &p = path[x][y][r][h];
25     bool &u = updated[x][y][r][h];
26
27     if( x < 0 || y < 0 )
28     {
29
30         return 0;
31     }
32
33     if( u == false )
34     {
35         const int
36             min_x = h + 2 * ( r - 1 ),
37             min_diff = min_x + 2 - h,
38             min_sum = min_x + h,
39             diff_xy = x - y,
40             sum_xy = x + y;
41         u = true;
```

```
42          if( diff_xy <= 0 )
43          {
44              p = diff_xy == 0 && !r;
45
46          }
47          else if( r && ( diff_xy < min_diff || sum_xy <
                min_sum ) )
48          {
49              p = 0;
50
51          }
52          else
53          {
54              if( y == h - 1 )
55              {
56                  if( r )
57                  {
58                      p += paths( x - 2, y, r - 1, h );
59
60                  }
61                  p += paths( x - 2, y + 2, r, h );
62
63              }
64              else
65              {
66                  p += paths( x - 1, y + 1, r, h );
67
68              }
69              if( y > 0 )
70              {
71                  p += paths( x - 1, y - 1, r, h );
72              }
73
74          }
75      }
76      else
77      {
78
79      }
80
81
82      return p;
83  }
84
85
86  int main ()
```

```
87 {
88     int n, r, k;
89
90     while( cin >> n >> r >> k, !cin.fail() )
91     {
92
93         cout << paths( n + n, 0, r, k ) << endl;
94
95     }
96
97     return 0;
98 }
```

A.9. Diagonal

```
1  #include <cstdio>
2  #include <cmath>
3
4  using namespace std;
5
6
7  typedef unsigned long long int uint;
8
9
10 inline int get_uint (uint &n)
11 {
12     return scanf ("%lu", &n);
13 }
14
15
16 inline uint min_sides (uint diagonals)
17 {
18     double d = (double)diagonals;
19     d = (3 + sqrt (9 + 8 * d)) / 2;
20     return (uint)ceil(d);
21 }
22
23
24 int main ()
25 {
26     int test_case = 0;
27     uint n;
28
29     while (get_uint (n), n != 0)
30     {
31         ++test_case;
32         printf("Case %d: %lu\n", test_case, min_sides (n
               ));
33     }
34
35     return 0;
36 }
```

A.10. Lonesome Knight

```
1  #include <iostream>
2
3  using namespace std;
4
5
6  const int board[8][8] = {
7      {2, 3,  4,  4,  4,  4,  3,  2},
8      {3, 4,  6,  6,  6,  6,  4,  3},
9      {4, 6,  8,  8,  8,  8,  6,  4},
10     {4, 6,  8,  8,  8,  8,  6,  4},
11     {4, 6,  8,  8,  8,  8,  6,  4},
12     {4, 6,  8,  8,  8,  8,  6,  4},
13     {3, 4,  6,  6,  6,  6,  4,  3},
14     {2, 3,  4,  4,  4,  4,  3,  2}
15 };
16
17
18 template <typename N>
19 inline void set_min (N &a, const N &b)
20 {
21     if (a > b)
22         a = b;
23 }
24
25
26 struct Knight
27 {
28     int f, c;
29
30     inline int attacked_squares () {
31         return board[f][c];
32     }
33 };
34
35
36 istream & operator >> (istream &s, Knight &k)
37 {
38     char c;
39     s >> c >> k.f;
40     k.c = c - 'a';
41     --k.f;
42 }
43
```

```
44
45 int main ()
46 {
47     Knight k;
48     int n;
49
50     cin >> n;
51     while (n--)
52     {
53         cin >> k;
54         cout << k.attacked_squares() << endl;
55     }
56
57     return 0;
58 }
```

A.11. Turn the Lights Off

```
1  #include <iostream>
2  #include <string>
3
4  using namespace std;
5
6  #define N 10
7  #define COUNT_MIN 0
8  #define COUNT_MAX 100
9  #define MAX_COMBINATIONS 1024
10 #define for_loop(i) for (uint i = 1; i <= N; ++i)
11
12 typedef unsigned int uint;
13
14 struct Cuad
15 {
16     bool c[N+2][N+2];
17
18     inline void flip (uint x, uint y) {
19         c[x][y] ^= 1;
20     }
21
22     inline void touch (uint x, uint y) {
23         flip (x, y);
24         flip (x-1, y);
25         flip (x+1, y);
26         flip (x, y-1);
27         flip (x, y+1);
28     }
29
30     inline int count () {
31         uint r = 0;
32         for_loop (i) {
33             for_loop (j)
34                 r += c[i][j];
35         }
36         return r;
37     }
38
39     inline bool uniform () {
40         uint r = count ();
41         return r == COUNT_MIN;
42     }
43 };
```

```
44
45 ostream & operator << (ostream &os, const Cuad c)
46 {
47     for_loop (x)
48     {
49         for_loop (y)
50         {
51             os << c.c[x][y];
52         }
53         os << '\n';
54     }
55     return os;
56 }
57
58 Cuad cuad;
59
60
61 inline uint bit (uint it, uint i)
62 {
63     return (it >> i) % 2;
64 }
65
66 inline void coord (uint i, uint &x, uint &y)
67 {
68     x = (i / N) + 1;
69     y = (i %N) + 1;
70 }
71
72
73 inline uint min_movements ()
74 {
75     uint min, moves, b, x, y;
76     Cuad aux;
77
78     if (cuad.uniform ())
79     {
80         min = 0;
81     }
82     else
83     {
84         min = COUNT_MAX + 1;
85         for (uint it = 0; it < MAX_COMBINATIONS; ++it)
86         {
87             aux = cuad;
88             moves = 0;
89             for (uint i = 0; i < N; ++i)
```

```
90              {
91                  b = bit (it, i);
92                  if (b) {
93                      ++moves;
94                      coord (i, x, y);
95                      aux.touch (x, y);
96                  }
97              }
98              for (x = 1; x < N; ++x)
99              {
100                 for (y = 1; y <= N; ++y)
101                 {
102                     if (aux.c[x][y])
103                     {
104                         ++moves;
105                         aux.touch (x + 1, y);
106                     }
107                 }
108             }
109             for (y = 1; y <= N; ++y)
110             {
111                 if (aux.c[x][y])
112                 {
113                     moves = COUNT_MAX + 1;
114                     break;
115                 }
116             }
117             if (min > moves)
118             {
119                 min = moves;
120             }
121         }
122     }
123
124     return min;
125 }
126
127
128 int main ()
129 {
130     int m;
131     char c;
132     string name;
133     while (cin >> name, name != "end")
134     {
135         for_loop (i)
```

```
136             {
137                 for_loop (j)
138                 {
139                     cin >> c;
140                     cuad.c[i][j] = (c == 'O');
141                 }
142             }
143 
144             m = min_movements ();
145 
146             if (m > COUNT_MAX)
147                 m = -1;
148 
149             cout << name << ' ' << m << endl;
150         }
151 
152         return 0;
153     }
```

A.12. Stars

```
1  #include <iostream>
2  #include <algorithm>
3
4  using namespace std;
5
6  #define MAX_N 15000
7
8
9  struct Star
10 {
11     int index, x, y;
12 };
13
14 istream & operator >> ( istream &is, Star &s )
15 {
16     return is >> s.x >> s.y;
17 }
18
19 ostream & operator << ( ostream &os, Star &s )
20 {
21     return os << s.x << ' ' << s.y;
22 }
23
24
25 inline bool less_x( const Star &s, const Star &t )
26 {
27     return s.x < t.x || ( s.x == t.x && s.y < t.y );
28 }
29
30 inline bool less_y( const Star &s, const Star &t )
31 {
32     return s.y < t.y || ( s.y == t.y && s.x < t.x );
33 }
34
35 inline bool operator < ( const Star &s, const Star &t )
36 {
37     return less_y( s, t );
38 }
39
40
41 struct Map
42 {
43     int level[MAX_N];
```

```
44
45      inline int & operator [] ( const Star &s ) {
46          return level[s.index];
47      }
48
49      inline int & operator [] ( const int i ) {
50          return level[i];
51      }
52  };
53
54
55  Map map;
56  Star star[MAXN];
57  int level[MAXN];
58
59
60  void calculate ( Star *vb, Star *ve )
61  {
62      int B;
63      const int n = ve - vb;
64      Star *b1, *e1, *b2, *e2, v_aux[n];
65
66
67      if( n > 1 )
68      {
69          for( b1 = vb, e2 = v_aux; b1 != ve; ++b1, ++e2 )
70          {
71              *e2 = *b1;
72          }
73          b1 = v_aux;
74          e1 = b2 = v_aux + ( n / 2 );
75          calculate( b1, e1 );
76          calculate( b2, e2 );
77          B = 0;
78          while( b1 != e1 && b2 != e2 )
79          {
80              if( *b2 < *b1 )
81              {
82                  map[*b2] += B;
83                  *vb++ = *b2++;
84              }
85              else
86              {
87                  ++B;
88                  *vb++ = *b1++;
89              }
```

```
90          }
91          while( b1 != e1 )
92          {
93              *vb++ = *b1++;
94          }
95          while( b2 != e2 )
96          {
97              map[*b2] += B;
98              *vb++ = *b2++;
99          }
100     }
101 }
102
103
104
105 int main ()
106 {
107     int n;
108
109     cin >> n;
110     for (int i = 0; i < n; ++i)
111     {
112         star[i].index = i;
113         cin >> star[i];
114     }
115
116     sort( star, star + n, less_x );
117     for (int i = 0; i < n; ++i)
118     {
119     }
120     calculate( star, star + n );
121
122     for (int i = 0; i < n; ++i)
123     {
124         ++level[ map[i] ];
125     }
126
127     for (int i = 0; i < n; ++i)
128     {
129         cout << level[i] << endl;
130     }
131
132     return 0;
133 }
```

A.13. Cyclic antimonotonic permutations

```
1  #include <iostream>
2  #include <algorithm>
3
4  using namespace std;
5
6  #define N 1000001
7
8  int list[N] = {3, 1};
9
10
11 inline void reset ()
12 {
13     list[2] = 2;
14 }
15
16
17 int main ()
18 {
19     int n, i;
20
21     while (cin >> n, n)
22     {
23         reset ();
24
25         for (i = 4; i <= n; ++i)
26         {
27             int &p0 = list[i-3];
28             int &p1 = list[i-2];
29             int &p2 = list[i-1];
30             p2 = i;
31             if (p0 > p1)
32                 swap (p0, p2);
33             else
34                 swap (p1, p2);
35         }
36
37         cout << list[0] << ' ' << list[1];
38         for (i = 2; i < n; ++i)
39             cout << ' ' << list[i];
40         cout << '\n';
41     }
42
43     return 0;
```

```
44  }
```

A.14. Ordering Tasks

```
1  #include <iostream>
2  #include <set>
3
4  using namespace std;
5
6  #define MAX_N 101
7
8  typedef set <int> Vector;
9  typedef Vector::iterator Iterator;
10
11 #define iterate(it,st) for (it = st.begin(); it != st.end
       (); ++it)
12
13 struct Node
14 {
15     Vector parent;
16     Vector child;
17 };
18
19
20 struct Tree
21 {
22     Node node[MAX_N];
23     int N;
24
25     inline void reset (int n = MAX_N) {
26         N = n;
27         for (int i = 1; i <= N; ++i)
28             node[i].parent.empty();
29     }
30
31     inline void connect (int i, int j) {
32         node[j].parent.insert (i);
33         node[i].child.insert (j);
34     }
35
36     inline void execute_task (int n) {
37         Iterator it;
38         iterate (it, node[n].child)
39         {
40             node[*it].parent.erase (n);
41         }
42     }
```

```
43
44      void list_dfs (int n) {
45          Iterator it;
46          cout << n << ' ';
47          iterate (it, node[n].child)
48          {
49              list_dfs (*it);
50          }
51      }
52
53      inline void list_tasks ()
54      {
55          Vector tasks;
56          Iterator it;
57          for (int i = 1; i <= N; ++i)
58          {
59              tasks.insert (i);
60          }
61
62          while (!tasks.empty())
63          {
64              iterate (it, tasks)
65              {
66                  if (node[*it].parent.empty())
67                  {
68                      cout << *it << ' ';
69                      execute_task (*it);
70                      tasks.erase (it);
71                      break;
72                  }
73              }
74          }
75      }
76  };
77
78  Tree tree;
79
80
81  int main ()
82  {
83      int m, n, i, j;
84
85      while (cin >> n >> m, m != 0 || n != 0)
86      {
87          tree.reset (n);
88          while (m--)
```

```
89          {
90              cin >> i >> j;
91              tree.connect (i, j);
92          }
93
94
95
96          tree.list_tasks ();
97          cout << endl;
98      }
99
100     return 0;
101 }
```

A.15. What Goes Up

```
1  #include <iostream>
2  #include <set>
3  #include <deque>
4  #include <list>
5
6  using namespace std;
7
8  #define NP -1
9
10
11 struct Pair
12 {
13     int first, second;
14 };
15
16 inline bool operator < ( const Pair &p, const Pair &q )
17 {
18     return p.first < q.first;
19 }
20
21 typedef deque <Pair> Sequence;
22 typedef list <int> Subsequence;
23 typedef set <Pair> Set;
24 typedef Set::iterator Iterator;
25
26
27 template <typename E>
28 ostream & operator << (ostream &os, const set <E> &s)
29 {
30     typename set <E>::const_iterator it;
31     os << '{';
32     if (! s.empty ())
33     {
34         it = s.begin ();
35         os << *it;
36         for (++it; it != s.end (); ++it)
37             os << ", " << *it;
38     }
39     os << '}';
40     return os;
41 }
42
43 template <typename E>
```

```
44 ostream & operator << (ostream &os, const deque <E> &s)
45 {
46     typename deque <E>::const_iterator it;
47     os << '{';
48     if (! s.empty ())
49     {
50         it = s.begin ();
51         os << *it;
52         for (++it; it != s.end (); ++it)
53             os << ", " << *it;
54     }
55     os << '}';
56     return os;
57 }
58
59 ostream & operator << (ostream &os, const Pair &p)
60 {
61     return os << '<' << p.first << ", " << p.second << '>
        ';
62 }
63
64
65 Sequence seq;
66 Subsequence l;
67
68
69 inline int lis ()
70 {
71     Set s;
72     pair <Iterator, bool> p_it;
73     Iterator &it = p_it.first;
74     int n = seq.size();
75     Pair p;
76     int &i = p.second;
77
78     for( i = 0; i < n; ++i )
79     {
80         p.first = seq[i].first;
81         p_it = s.insert( p );
82         if( it != s.begin() )
83         {
84             Iterator it_aux = it;
85             --it_aux;
86             seq[i].second = (*it_aux).second;
87         }
88         if(p_it.second && ++it != s.end() )
```

```
89              s.erase( it );
90
91      }
92
93      p = *( s.rbegin() );
94      do
95      {
96          p = seq[i];
97          l.push_front (p.first);
98
99      }
100     while (i != NP);
101
102     return s.size();
103 }
104
105
106 int main ()
107 {
108     Pair p;
109     Subsequence::iterator it;
110
111     p.second = NP;
112     while( !cin.eof() )
113     {
114         cin >> p.first;
115         cin.ignore();
116         seq.push_back( p );
117     }
118
119     cout << lis() << "\n-\n";
120     for( it = l.begin(); it != l.end(); ++it )
121     {
122         cout << *it << '\n';
123     }
124
125     return 0;
126 }
```

A.16. The primary problem

```
1  #include <cstdio>
2  #include <cmath>
3
4  #define MAX_N 1000000
5
6
7  int PL = 2,
8      prime_list[MAX_N] = {2, 3};
9  bool is_prime[MAX_N+6] = {false, false, true, true};
10
11
12 inline void init ()
13 {
14     int p, i;
15     const int SQN = (int) sqrt ((double)MAX_N);
16
17     for (p = 1; p <= MAX_N; p += 6)
18     {
19         is_prime[p] = true;
20         is_prime[p + 4] = true;
21     }
22
23     for (p = 5; p <= MAX_N; ++p)
24     {
25         while (is_prime[p] == false)
26         {
27             ++p;
28             if (p > MAX_N)
29                 return;
30         }
31         prime_list[PL++] = p;
32         if (p <= SQN)
33         {
34             for (i = p * p; i <= MAX_N; i += p)
35             {
36                 is_prime[i] = false;
37             }
38         }
39     }
40 }
41
42
43 int main ()
```

```
44 {
45     int n, p1, p2;
46     init ();
47     while (scanf("%d", &n), n)
48     {
49         printf("%d:\n", n);
50         int n2 = n / 2;
51         p1 = prime_list[0];
52         for (int i = 0; p1 <= n2; ++i, p1 = prime_list[i
               ])
53         {
54             p2 = n - p1;
55             if (is_prime[p2])
56             {
57                 printf("%d+%d\n", p1, p2);
58                 break;
59             }
60         }
61         if (p1 > n2)
62             puts("NO_WAY!");
63     }
64 }
```

A.17. Minimal coverage

```
1  #include <iostream>
2  #include <algorithm>
3
4  using namespace std;
5
6  #define MAX_SEGMENTS 100000
7
8  typedef int stype;
9
10
11 template <typename N>
12 inline void set_max (N &a, const N &b)
13 {
14     if (a < b)
15         a = b;
16 }
17
18
19 struct Segment
20 {
21     stype start, finish;
22 };
23
24
25 inline bool operator < (const Segment &a, const Segment &
       b)
26 {
27     return (a.start < b.start) || (a.start == b.start &&
           a.finish > b.finish);
28 }
29
30 istream & operator >> (istream &s, Segment &a)
31 {
32     return s >> a.start >> a.finish;
33 }
34
35 ostream & operator << (ostream &s, Segment &a)
36 {
37     return s << a.start << ' ' << a.finish;
38 }
39
40
41 Segment segment[MAX_SEGMENTS];
```

```
42 Segment choosed[MAX_SEGMENTS];
43 stype minimum, maximum;
44 int n;
45
46
47 inline int min_subset ()
48 {
49     int count, i;
50     Segment f;
51
52     sort (segment, segment + n);
53     if (segment[0].start > minimum)
54         return 0;
55
56     minimum = 0;
57     count = 0;
58     for (i = 0; i < n;)
59     {
60         f.finish = minimum;
61         for (; i < n && segment[i].start <= minimum; ++i)
62         {
63             if (f.finish < segment[i].finish) {
64                 f = segment[i];
65             }
66         }
67         if (f.finish <= minimum)
68             return 0;
69         choosed[count] = f;
70         ++count;
71         if (f.finish >= maximum)
72             return count;
73         minimum = f.finish;
74     }
75
76     return 0;
77 }
78
79
80 int main ()
81 {
82     int test_cases, subset;
83     Segment seg;
84
85     cin >> test_cases;
86     while (test_cases--)
87     {
```

```
88          cin >> maximum;
89          n = 0;
90          while (cin >> seg, seg.start || seg.finish)
91          {
92              if (seg.finish <= 0)
93                  continue;
94              segment[n] = seg;
95              ++n;
96          }
97
98          subset = min_subset ();
99          cout << subset << endl;
100         for (int i = 0; i < subset; ++i)
101         {
102             cout << choosed[i] << endl;
103         }
104         cout << endl;
105     }
106
107     return 0;
108 }
```

A.18. Prime Cuts

```
1  #include <cstdio>
2
3  #define MAX_N 1001
4
5
6  int total_primes_until[MAX_N] = {0,1,2,3};
7  int prime_list[MAX_N] = {1,2,3};
8
9
10 int prime (size_t pos)
11 {
12     bool prime_not_found;
13
14     int &p = prime_list[pos];
15     if (p)
16         return p;
17
18     p = prime(pos-1) + 2;
19     prime_not_found = true;
20     while (prime_not_found)
21     {
22         prime_not_found = false;
23         for (int *pp = prime_list + 1; pp < &p; ++pp)
24         {
25             if (p %*pp == 0)
26             {
27                 prime_not_found = true;
28                 p += 2;
29                 break;
30             }
31         }
32     }
33
34     const int &tp = total_primes_until[prime_list[pos
           -1]];
35     const int limit = p < MAX_N ? p : MAX_N;
36     for (int i = prime_list[pos - 1] + 1; i < limit; ++i)
37         total_primes_until[i] = tp;
38     if (p < MAX_N)
39         total_primes_until[p] = tp + 1;
40
41     return p;
42 }
```

```
43
44
45  int main ()
46  {
47      int n, c, t, ini, fin;
48
49      prime (1000);
50      /*for (int i = 0; i < MAX_N; ++i)
51          fprintf(stderr, "%4d: %d\n", i, prime(i));*/
52
53      while (scanf ("%d %d", &n, &c) == 2)
54      {
55          printf ("%d %d:", n, c);
56
57          t = total_primes_until[n];
58          ini = (t + 1) / 2 - c;
59          if (ini < 0)
60              ini = 0;
61          fin = t - ini;
62
63          for (int i = ini; i < fin; ++i)
64              printf(" %d", prime_list[i]);
65          putchar ('\n');
66          putchar ('\n');
67      }
68
69      return 0;
70  }
```

A.19. Median on the Plane

```
1  #include <iostream>
2  #include <cmath>
3  #include <algorithm>
4  using namespace std;
5
6  #define MAX_N 10000
7
8  typedef double coord;
9  typedef double angle;
10
11 const angle PI = atan( 1.0 ) * 4,
12     MITAD_PI = PI / 2;
13
14
15 struct Punto
16 {
17     coord x, y;
18     angle a;
19     int index;
20 };
21
22
23 bool operator < ( const Punto &A, const Punto &B )
24 {
25     return ( A.y == B.y ) ? ( A.x < B.x ) : ( A.y < B.y )
           ;
26 }
27
28
29 bool comp_angulo( const Punto &A, const Punto &B )
30 {
31     return A.a < B.a;
32 }
33
34
35 istream & operator >> ( istream &is, Punto &P )
36 {
37     return is >> P.x >> P.y;
38 }
39
40
41 ostream & operator << ( ostream &os, const Punto &P )
42 {
```

```
43      return os << (P.index + 2) << ". (" << P.x << "," <<
            P.y << ')';
44  }
45
46
47  inline angle angulo( const Punto &A, const Punto &B )
48  {
49      Punto P;
50      angle a;
51      P.x = B.x - A.x;
52      P.y = B.y - A.y;
53      if( P.x == 0 )
54      {
55          return P.y > 0 ? MITAD_PI : - MITAD_PI;
56      }
57      a = atan( P.y / P.x );
58      return a < 0 ? MITAD_PI - a : a;
59  }
60
61
62  int N;
63  Punto P[MAX_N], P0;
64
65
66  inline void insertar( const Punto &Pi, int i )
67  {
68      if( Pi < P0 )
69      {
70          P[i] = P0;
71          P0 = Pi;
72      }
73      else
74      {
75          P[i] = Pi;
76      }
77  }
78
79
80  int main ()
81  {
82      Punto Q;
83      int M;
84
85      cin >> N;
86      --N;
87
```

```
88      cin >> P0;
89      P0.index = -1;
90      for( int i = 0; i < N; ++i )
91      {
92          cin >> Q;
93          Q.index = i;
94          insertar( Q, i );
95      }
96
97
98
99      for( int i = 0; i < N; ++i )
100     {
101         P[i].a = angulo( P0, P[i] );
102
103     }
104     sort( P, P + N, comp_angulo );
105
106
107
108     M = N / 2;
109     cout << ( P0.index + 2 ) << ' ' << ( P[M].index + 2 )
            ;
110
111     return 0;
112 }
```

A.20. Rabbit Hunt

```
1  #include <iostream>
2  #include <map>
3  #include <set>
4
5  using namespace std;
6
7  #define MAX_RABBITS 200
8  #define SPECIAL 10000.0
9
10 struct Point
11 {
12     double x, y;
13 };
14
15 bool operator == (const Point &a, const Point &b)
16 {
17     return a.x == b.x && a.y == b.y;
18 }
19
20 bool operator < (const Point &a, const Point &b)
21 {
22     return a.x < b.x || (a.x == b.x && a.y < b.y);
23 }
24
25 istream & operator >> (istream &is, Point &p)
26 {
27     return is >> p.x >> p.y;
28 }
29
30 ostream & operator << (ostream &os, const Point &p)
31 {
32     return os << p.x << ' ' << p.y;
33 }
34
35 typedef map <Point, set <Point> > HashTable;
36
37
38 inline void pendiente (const Point &u, const Point &v,
       Point &m)
39 {
40     m.x = u.x == v.x
41         ? SPECIAL
42         : (v.y - u.y) / (v.x - u.x);
```

```
43      m.y = -m.x * u.x + u.y;
44  }
45
46
47  Point rabbit[MAX_RABBITS];
48  HashTable conteo;
49
50
51  int main ()
52  {
53      int n, r, c;
54      Point m;
55
56      cin >> n;
57      for (int i = 0; i < n; ++i)
58      {
59          cin >> rabbit[i];
60      }
61
62      for (int i = 0; i < n; ++i)
63      {
64          for (int j = i + 1; j < n; ++j)
65          {
66              pendiente (rabbit[i], rabbit[j], m);
67              //cerr << "\t\t" << rabbit[i] << '\t' <<
                    rabbit[j] << '\t' << m << endl;
68              conteo[m].insert (rabbit[i]);
69              conteo[m].insert (rabbit[j]);
70          }
71      }
72
73      r = 0;
74      for (HashTable::iterator it = conteo.begin(); it !=
            conteo.end(); ++it)
75      {
76          //cerr << '\t' << it->first << '\t' << it->second
                << endl;
77          c = it->second.size();
78          if (r < c)
79              r = c;
80      }
81      cout << r << endl;
82
83      return 0;
84  }
```

A.21. Ecological Premium

```
1  #include <iostream>
2  using namespace std;
3
4  int main ()
5  {
6      int test_cases, farmers, size, animals,
           environment_friendliness, sum;
7
8      cin >> test_cases;
9
10     while (test_cases--)
11     {
12         cin >> farmers;
13         sum = 0;
14         while (farmers--)
15         {
16             cin >> size >> animals >>
                   environment_friendliness;
17             sum += size * environment_friendliness;
18         }
19         cout << sum << endl;
20     }
21
22     return 0;
23 }
```

A.22. Funny Game

```
1  #include <iostream>
2  #include <set>
3  using namespace std;
4
5  #define MAX_N 1001
6  #define PERDEDOR 0
7
8  typedef set <size_t> Set;
9  typedef Set::iterator Iterator;
10
11
12 ostream & operator << ( ostream &os, const Set &s )
13 {
14     Iterator it;
15     os << '{';
16     if( s.size() > 0 )
17     {
18         it = s.begin();
19         os << *it;
20         for( ++it; it != s.end(); ++it )
21         {
22             os << ", " << *it;
23         }
24     }
25     return os << '}';
26 }
27
28
29 struct Aeropuerto
30 {
31     Set conexion;
32
33
34     inline void conectar( size_t i ) {
35         conexion.insert( i );
36     }
37
38     inline void desconectar( size_t i ) {
39         conexion.erase( i );
40     }
41 };
42
43
```

```
44 struct Grafo
45 {
46     Aeropuerto A[MAX_N];
47
48
49     inline void conectar( size_t i, size_t j ) {
50         A[i].conectar( j );
51         A[j].conectar( i );
52     }
53
54     inline void explotar( size_t i ) {
55         Iterator it;
56         Set &c = A[i].conexion;
57         for( it = c.begin(); it != c.end(); ++it ) {
58             A[*it].desconectar( i );
59         }
60     }
61
62     inline void restaurar( size_t i ) {
63         Iterator it;
64         Set &c = A[i].conexion;
65         for( it = c.begin(); it != c.end(); ++it ) {
66             A[*it].conectar( i );
67         }
68     }
69
70
71     size_t jugar( size_t i );
72 };
73
74
75 size_t Grafo::jugar( size_t i )
76 {
77     Iterator it;
78     Set &c = A[i].conexion;
79     size_t r = PERDEDOR;
80
81     if( c.size() > 0 )
82     {
83         explotar( i );
84         for( it = c.begin(); it != c.end(); ++it )
85         {
86             if( jugar( *it) == PERDEDOR )
87             {
88                 r = *it;
89                 break;
```

```
90              }
91          }
92          restaurar( i );
93      }
94
95      return r;
96  }
97
98
99  Grafo grafo;
100
101
102 int main ()
103 {
104     size_t n, k, o, d;
105
106     cin >> n >> k;
107     for( size_t i = 1; i < n; ++i )
108     {
109         cin >> o >> d;
110         grafo.conectar( o, d );
111     }
112     n = grafo.jugar( k );
113     if( n == PERDEDOR )
114         cout << "First player loses";
115     else
116         cout << "First player wins flying to airport " <<
                n;
117
118     return 0;
119 }
```

A.23. Flip Game

```
1  #include <iostream>
2
3  using namespace std;
4
5  #define N 4
6  #define COUNT_MIN 0
7  #define COUNT_MAX 16
8  #define MAX_COMBINATIONS 65536
9  #define for_loop(i) for (uint i = 1; i <= N; ++i)
10
11 typedef unsigned int uint;
12
13 struct Cuad
14 {
15     bool c[N+2][N+2];
16
17     inline void flip (uint x, uint y) {
18         c[x][y] ^= 1;
19     }
20
21     inline void touch (uint x, uint y) {
22         flip (x, y);
23         flip (x-1, y);
24         flip (x+1, y);
25         flip (x, y-1);
26         flip (x, y+1);
27     }
28
29     inline int count () {
30         uint r = 0;
31         for_loop (i) {
32             for_loop (j)
33                 r += c[i][j];
34         }
35         return r;
36     }
37
38     inline bool uniform () {
39         uint r = count ();
40         return r == COUNT_MIN || r == COUNT_MAX;
41     }
42 };
43
```

```
44 Cuad cuad;
45
46
47 inline uint bit (uint it, uint i)
48 {
49     return (it >> i) % 2;
50 }
51
52 inline void coord (uint i, uint &x, uint &y)
53 {
54     x = (i / N) + 1;
55     y = (i %N) + 1;
56 }
57
58
59 inline uint min_movements ()
60 {
61     uint min, moves, b, x, y;
62     Cuad aux;
63
64     if (cuad.uniform ())
65     {
66         min = 0;
67     }
68     else
69     {
70         min = COUNT_MAX + 1;
71         for (uint it = 0; it < MAX_COMBINATIONS; ++it)
72         {
73             aux = cuad;
74             moves = 0;
75             for (uint i = 0; i < COUNT_MAX; ++i)
76             {
77                 b = bit (it, i);
78                 if (b) {
79                     ++moves;
80                     coord (i, x, y);
81                     aux.touch (x, y);
82                 }
83             }
84             if (min > moves && aux.uniform ())
85             {
86                 min = moves;
87             }
88         }
89     }
```

```
90
91      return min;
92  }
93
94
95  int main ()
96  {
97      uint m;
98      char c;
99      for_loop (i)
100     {
101         for_loop (j)
102         {
103             cin >> c;
104             cuad.c[i][j] = (c == 'b');
105         }
106     }
107
108     m = min_movements ();
109
110     if (m > COUNT_MAX)
111         cout << "Impossible\n";
112     else
113         cout << m << endl;
114
115     return 0;
116 }
```

A.24. Longest path in a tree

```
1  #include <iostream>
2  #include <vector>
3
4  using namespace std;
5
6  #define MAX_N 10001
7
8  #define for_edge(i,l) for (int i = 1; i < l; ++i)
9
10 typedef vector <int> Children;
11
12
13 struct Node
14 {
15     Children child;
16
17     inline void add_child (int c) {
18         child.push_back (c);
19     }
20 };
21
22
23 struct Graph
24 {
25     Node node[MAX_N];
26     bool marked[MAX_N];
27
28     inline int connect (int i, int j) {
29         node[i].add_child (j);
30         node[j].add_child (i);
31     }
32
33     int longest_path (int n)
34     {
35         int c, s, max, l;
36         Children &ch = node[n].child;
37
38         max = -1;
39         marked[n] = true;
40         s = ch.size();
41         for (int i = 0; i < s; ++i)
42         {
43             c = ch[i];
```

```
44              if (marked[c])
45                  continue;
46              l = longest_path (c);
47              if (max < l)
48                  max = l;
49          }
50          marked[n] = false;
51          return max + 1;
52      }
53
54      int longest_bipath (int n)
55      {
56          int c, s, max[2], l;
57          Children &ch = node[n].child;
58
59          max[0] = max[1] = -1;
60          marked[n] = true;
61          s = ch.size();
62          for (int i = 0; i < s; ++i)
63          {
64              c = ch[i];
65              if (marked[c])
66                  continue;
67              l = longest_path (c);
68              if (max[0] < l) {
69                  max[1] = max[0];
70                  max[0] = l;
71              }
72              else if (max[1] < l) {
73                  max[1] = l;
74              }
75          }
76          return max[0] + max[1] + 2;
77      }
78
79      int dfs (int n, int &z)
80      {
81          int c, s, max, l, aux;
82          Children &ch = node[n].child;
83
84          max = -1;
85          marked[n] = true;
86          s = ch.size();
87          for (int i = 0; i < s; ++i)
88          {
89              c = ch[i];
```

```
90              if (marked[c])
91                  continue;
92              l = dfs (c, aux);
93              if (max < l)
94              {
95                  max = l;
96                  z = aux;
97              }
98          }
99          if (max == -1)
100             z = n;
101         marked[n] = false;
102         return max + 1;
103     }
104 };
105
106
107 Graph graph;
108
109
110 int main ()
111 {
112     int n, u, v;
113
114     cin >> n;
115     if (n == 1)
116         cout << 0;
117     else
118     {
119         while (--n > 0)
120         {
121             cin >> u >> v;
122             graph.connect (u, v);
123         }
124
125         graph.dfs (u, v);
126
127         cout << graph.dfs (v, u);
128     }
129     cout << endl;
130
131     return 0;
132 }
```

A.25. Tornado!

```
1  #include <iostream>
2
3  using namespace std;
4
5  #define N 5000
6
7  bool post[2*N];
8
9
10 inline int wooden_posts (int n)
11 {
12     int r, c, i = 0;
13     r = 0;
14     while (post[i] == false && i < n)
15         ++i;
16     if (i >= n)
17         return (n - 1) / 2 + 1;
18     while (i < n)
19     {
20         ++i;
21         c = 0;
22         while (post[i] == false)
23         {
24             ++i;
25             ++c;
26         }
27         r += c / 2;
28     }
29     return r;
30 }
31
32
33 int main ()
34 {
35     int n;
36     bool p;
37
38     while (cin >> n, n)
39     {
40         for (int i = 0; i < n; ++i)
41         {
42             cin >> p;
43             post[i] = post[i + n] = p;
```

```
44            }
45
46            cout << wooden_posts (n) << endl;
47        }
48
49        return 0;
50    }
```

A.26. Product of Digits

```
1  #include <iostream>
2
3  using namespace std;
4
5  #define MAX_DIGITS 10
6
7  typedef unsigned int uint;
8
9  uint digit_count[MAX_DIGITS];
10
11
12 int main ()
13 {
14     uint n, d, dc;
15     char c;
16
17     cin >> n;
18
19     if (n == 1)
20         cout << n << endl;
21     else if (n == 0)
22         cout << 10 << endl;
23     else
24     {
25         for (d = MAX_DIGITS - 1; n > 1 && d >= 2; )
26         {
27             if (n %d)
28                 --d;
29             else
30             {
31                 ++digit_count[d];
32                 n /= d;
33             }
34         }
35         if (n > 1)
36             cout << -1 << endl;
37         else
38         {
39             for (d = 2; d < MAX_DIGITS; ++d)
40             {
41                 dc = digit_count[d];
42                 c = d + '0';
43                 while (dc--)
```

```
44                          cout << c;
45                  }
46                  cout << '\n';
47              }
48      }
49
50      return 0;
51  }
```

A.27. Two Teams

```
1  #include <iostream>
2  #include <set>
3  using namespace std;
4
5  #define MAX_N 101
6  #define UNKNOWN 0
7  typedef set <int> Set;
8  typedef Set::iterator Iterator;
9
10
11 inline void switch_team ( int &t )
12 {
13     t = 3 - t;
14 }
15
16
17 struct Node
18 {
19     int team;
20     Set friends;
21
22     inline void add_friend( int f ) {
23         friends.insert( f );
24     }
25
26     inline void delete_friend( int f ) {
27         friends.erase( f );
28     }
29 };
30
31
32 struct Graph
33 {
34     Node member[MAX_N];
35     int N;
36
37     inline void add_friend( int i, int j ) {
38         member[i].add_friend( j );
39     }
40
41
42     void set_teams( int i, int t )
43     {
```

```
44          int j;
45          Iterator it;
46          Node &mi = member[i];
47
48          mi.team = t;
49          switch_team( t );
50          while( !mi.friends.empty() )
51          {
52              it = mi.friends.begin();
53              j = *it;
54              Node &mj = member[j];
55              mi.friends.erase( it );
56              mj.delete_friend( i );
57              if( mj.team == UNKNOWN )
58                  set_teams( j, t );
59          }
60      }
61
62      inline int calculate_teams()
63      {
64          int count = 0;
65          for( int i = 1; i <= N; ++i )
66          {
67              switch( member[i].team )
68              {
69                  case UNKNOWN:
70                      set_teams( i, 1 );
71                  case 1:
72                      ++count;
73              }
74          }
75          return count;
76      }
77  };
78
79  Graph graph;
80
81
82  int main ()
83  {
84      int f;
85      cin >> graph.N;
86      for( int i = 1; i <= graph.N; ++i )
87      {
88          cin >> f;
89          if( f == 0 )
```

```
90          {
91              cout << 0 << endl;
92              return 0;
93          }
94          do
95          {
96              graph.add_friend( i, f );
97              cin >> f;
98          }
99          while( f );
100     }
101     cout << graph.calculate_teams() << endl;
102     cout << 1;
103     for (int i = 2; i <= graph.N; ++i)
104     {
105         if( graph.member[i].team == 1 )
106             cout << ' ' << i;
107     }
108     cout << endl;
109
110     return 0;
111 }
```

A.28. Basic wall maze

```
1  #include <iostream>
2  #include <queue>
3  #include <string>
4
5  using namespace std;
6
7  #define N 6
8  #define W (N + 1)
9  #define ROWS (N + W)
10 #define COLUMS ROWS
11 #define ADYACENCIAS 4
12 #define WALLS 3
13
14
15 struct Square
16 {
17     int r, c;
18
19     inline void operator += (const Square &s) {
20         r += s.r;
21         c += s.c;
22     }
23
24     inline bool is_invalid () {
25         return r < 0 || c < 0 || r >= ROWS || c >= COLUMS
               ;
26     }
27
28     inline void set_sum (const Square &a, const Square &b
           ) {
29         operator = (a);
30         operator += (b);
31     }
32
33     inline void to_wall () {
34         r *= 2;
35         c *= 2;
36     }
37
38     inline void to_grid () {
39         to_wall ();
40         --r;
41         --c;
```

```
42      }
43  };
44
45  istream & operator >> (istream &is, Square &sq)
46  {
47      return is >> sq.r >> sq.c;
48  }
49
50  inline bool operator == (const Square &a, const Square &b
        )
51  {
52      return a.r == b.r && a.c == b.c;
53  }
54
55  inline bool operator != (const Square &a, const Square &b
        )
56  {
57      return a.r != b.r || a.c != b.c;
58  }
59
60
61  const Square ady[ADYACENCIAS] = {
62      { 0, -2},
63      {-2,  0},
64      { 0,  2},
65      { 2,  0}
66  };
67
68  const char dir[ADYACENCIAS] = {
69      'N',
70      'W',
71      'S',
72      'E'
73  };
74
75
76  struct Path
77  {
78      Square position;
79      string path;
80
81      inline void push (const char d, const Square &s) {
82          position = s;
83          path += d;
84      }
85
```

```
86      inline void push (const char d) {
87          path += d;
88      }
89
90      inline void go (int d) {
91          position += ady[d];
92          path += dir[d];
93      }
94  };
95
96
97  struct Grid
98  {
99      bool board[COLUMS][ROWS];
100
101     Grid () : board() {}
102
103
104     inline void add_wall (const Square &ws, const Square
            &we)
105     {
106         int r, c;
107         r = ws.r;
108         c = ws.c;
109         if (r == we.r)
110         {
111             for (; c <= we.c; ++c)
112             {
113                 board[c][r] = true;
114             }
115         }
116         else
117         {
118             for (; r <= we.r; ++r)
119             {
120                 board[c][r] = true;
121             }
122         }
123     }
124
125
126     inline bool & at (const Square &sq)
127     {
128         return board[sq.c][sq.r];
129     }
130
```

```
131
132     inline bool & wall_between (const Square &a, const
            Square &b)
133     {
134         int r, c;
135         r = (a.r + b.r) / 2;
136         c = (a.c + b.c) / 2;
137         return board[c][r];
138     }
139
140
141     inline string path (const Square &start, const Square
            &end)
142     {
143         Path p, p_aux;
144         Square sq;
145         queue <Path> q;
146
147         p.position = start;
148         while (p.position != end)
149         {
150             at (p.position) = true;
151             for (int i = 0; i < ADYACENCIAS; ++i)
152             {
153                 p_aux = p;
154                 p_aux.go (i);
155                 if (p_aux.position.is_invalid ())
156                     continue;
157                 if (at (p_aux.position))
158                     continue;
159                 if (wall_between (p.position, p_aux.
                        position))
160                     continue;
161                 q.push (p_aux);
162             }
163
164             p = q.front (); q.pop ();
165         }
166
167         return p.path;
168     }
169 };
170
171
172 ostream & operator << (ostream &os, const Grid &g)
173 {
```

```
174     for (int c = 0; c < COLUMS; ++c)
175     {
176         for (int r = 0; r < ROWS; ++r)
177         {
178             os << g.board[c][r];
179         }
180         os << '\n';
181     }
182     return os;
183 }
184
185
186 int main ()
187 {
188     Square start, end, ws, we;
189
190     while (cin >> start, start.r || start.c)
191     {
192         Grid grid;
193         cin >> end;
194         start.to_grid ();
195         end.to_grid ();
196         for (int i = 0; i < WALLS; ++i)
197         {
198             cin >> ws >> we;
199             ws.to_wall ();
200             we.to_wall ();
201             grid.add_wall (ws, we);
202         }
203         cout << grid.path (start, end) << endl;
204     }
205
206     return 0;
207 }
```

A.29. Rope

```
1  #include <iostream>
2  #include <iomanip>
3  #include <cmath>
4
5  using namespace std;
6
7  #define MAX_N 100
8
9  const double PI = 3.14159265358979323846264338327 95;
10
11
12 struct Punto
13 {
14     double x, y;
15 };
16
17
18 istream & operator >> (istream &s, Punto &P)
19 {
20     return s >> P.x >> P.y;
21 }
22
23
24 inline double distancia (const Punto &A, const Punto &B)
25 {
26     double x, y;
27     x = A.x - B.x;
28     y = A.y - B.y;
29     x *= x;
30     y *= y;
31     return sqrt (x + y);
32 }
33
34
35 Punto P[MAX_N];
36
37
38 int main ()
39 {
40     int n;
41     double r, d;
42
43     cin >> n >> r;
```

```
44
45      cin >> P[0];
46      d = 0;
47      for (int i = 1; i < n; ++i)
48      {
49          cin >> P[i];
50          d += distancia (P[i], P[i-1]);
51      }
52      d += distancia (P[0], P[n-1]);
53      d += PI * r * 2;
54
55      cout << fixed << setprecision (2) << d << endl;
56  }
```

A.30. Binary Lexicographic Sequence

```
1  #include <iostream>
2
3  using namespace std;
4
5  #define MAX_N 44
6  #define MAX_K 1000000000
7
8
9  struct Range
10 {
11     int ini, fin;
12 };
13
14
15 Range range[MAX_N] = {
16     {1, 1},
17     {2, 2}
18     // Fill with init ().
19 };
20
21
22 ostream & operator << (ostream &os, const Range &r)
23 {
24     return os << r.ini << ' ' << r.fin;
25 }
26
27
28 inline void init ()
29 {
30     Range *r, *last;
31     last = range + MAX_N;
32     for (r = range + 2; r < last; ++r)
33     {
34         r->ini = r[-1].fin + 1;
35         r->fin = r[-1].fin + r[-2].fin;
36     }
37 }
38
39
40 int main ()
41 {
42     int n, k;
43
```

```
44      init ();
45
46      cin >> n >> k;
47
48      if (k > range[n].fin)
49      {
50          cout << -1;
51      }
52      else
53      {
54          while (n)
55          {
56              if (range[n].ini > k)
57              {
58                  cout << 0;
59              }
60              else
61              {
62                  cout << 1;
63                  k -= range[n-1].fin;
64              }
65              --n;
66          }
67      }
68
69      cout << endl;
70
71      return 0;
72  }
```

A.31. Distinct Subsequences

```
1  #include <iostream>
2  #include <cctype>
3
4  using namespace std;
5
6  #define MOD 1000000007
7  #define FIRST ('A' - 1)
8  #define LAST 'Z'
9  #define LETTERS (LAST - FIRST + 1)
10
11
12 struct Sub
13 {
14     int count, previous;
15 };
16
17
18 inline int distinct_subsequences()
19 {
20     Sub sub[LETTERS] = {{1, 0}}, old;
21     int c_old, c;
22     char ch;
23     c = 0;
24     while( ( cin.get( ch ), isupper( ch ) ) && !cin.eof()
           )
25     {
26         c_old = c;
27         c = ch - FIRST;
28         old = sub[c_old];
29         //sub[c].count = 2 * old.count;
30         sub[c].count = ( 2 * old.count ) %MOD;
31         if( sub[c].count )
32         {
33             //sub[c].count -= sub[c].previous;
34             sub[c].count += MOD - sub[c].previous;
35             sub[c].count %= MOD;
36         }
37         sub[c].previous = old.count;
38     }
39     return sub[c].count;
40 }
41
42 int main ()
```

```
43 {
44     int test_cases;
45     char c;
46 
47     cin >> test_cases;
48     cin.ignore();
49     while (test_cases--)
50     {
51         cout << distinct_subsequences() << endl;
52     }
53 
54     return 0;
55 }
```

A.32. Edit distance

```
1  #include <iostream>
2  #include <algorithm>
3  #include <cctype>
4  #include <cstring>
5  using namespace std;
6
7  #define MAX_LENGTH 2001
8  #define UNKNOWN 2002
9  typedef unsigned int uint;
10
11 struct String
12 {
13     char str[MAX_LENGTH];
14     uint len;
15
16     inline char & operator [] (uint n) {return str[n];}
17 };
18
19 istream & operator >> (istream &is, String &s)
20 {
21     is >> s.str;
22     s.len = strlen( s.str );
23     return is;
24 }
25
26
27 struct Strings
28 {
29     String str_a, str_b;
30     uint ED[ MAX_LENGTH ][ MAX_LENGTH ];
31
32     inline void reset ()
33     {
34         for (uint a = 0; a < str_a.len; ++a)
35         {
36             for (uint b = 0; b < str_b.len; ++b)
37             {
38                 ED[a][b] = UNKNOWN;
39             }
40         }
41     }
42
43
```

```
44      uint edit_distance (uint a, uint b)
45      {
46          if( !str_a[a] )
47              return str_b.len - b;
48          if( !str_b[b] )
49              return str_a.len - a;
50
51          uint &c = ED[a][b];
52          if (c == UNKNOWN)
53          {
54              c = edit_distance( a + 1, b + 1 );
55              if (str_a[a] != str_b[b])
56              {
57                  uint ma, mb;
58                  ma = edit_distance( a + 1, b );
59                  mb = edit_distance( a, b + 1 );
60                  c = min( min( ma, mb ), c ) + 1;
61              }
62          }
63          return c;
64      }
65
66
67      inline uint edit_distance ()
68      {
69          return edit_distance( 0, 0 );
70      }
71  };
72
73
74  istream & operator >> (istream &is, Strings &s)
75  {
76      is >> s.str_a >> s.str_b;
77      s.reset();
78      return is;
79  }
80
81
82  Strings s;
83
84
85  int main ()
86  {
87      uint t;
88      cin >> t;
89      while (t--)
```

```
90      {
91          cin >> s;
92          cout << s.edit_distance() << endl;
93      }
94      return 0;
95  }
```

A.33. Cutting Sticks

```
1  #include <iostream>
2  using namespace std;
3
4
5  struct Vector
6  {
7      int *begin, *end;
8
9      inline int & operator [] (size_t i) {return begin[i
           ];}
10     inline int operator [] (size_t i) const {return begin
           [i];}
11
12     inline int size () const
13     {
14         return end - begin;
15     }
16 };
17
18 bool operator < (const Vector &v1, const Vector &v2)
19 {
20     int s1, s2;
21     s1 = v1.size();
22     s2 = v2.size();
23     if( s1 == s2 )
24     {
25         const int *i1, *i2;
26         i1 = v1.begin;
27         i2 = v2.begin;
28         while( i1 != v1.end && *i1 == *i2 )
29         {
30             ++i1;
31             ++i2;
32         }
33         return i1 != v1.end && *i1 < *i2;
34     }
35     else
36     {
37         return s1 < s2;
38     }
39 }
40
41
```

```
42 #define MAX_LENGTH 1001
43 #define MAX_CUTS 52
44 #define MAX_COST (MAX_CUTS * MAX_LENGTH)
45 #define UNKNOWN MAX_COST
46
47 int cuts[MAX_CUTS], cost[MAX_LENGTH][MAX_LENGTH];
48 Vector vec = {cuts};
49
50 inline void reset (int length)
51 {
52     vec.end = cuts;
53     for( int i = 0; i <= length; ++i )
54     {
55         for( int j = 0; j <= length; ++j )
56         {
57             cost[i][j] = UNKNOWN;
58         }
59     }
60 }
61
62
63 int min_cost (const Vector &v)
64 {
65     int &begin = *v.begin,
66         &end = *v.end,
67         &cost_value = cost[begin][end];
68     if( cost_value == UNKNOWN )
69     {
70         int cost_base, cost_aux;
71         cost_base = end - begin;
72         switch( v.size() - 1 )
73         {
74             case 0:
75             {
76                 cost_value = 0;
77                 break;
78             }
79             case 1:
80             {
81                 cost_value = cost_base;
82                 break;
83             }
84             default:
85             {
86                 Vector v1, v2;
87                 v1.begin = v.begin;
```

```
88                  v2.end = v.end;
89                  cost_value = MAX_COST;
90                  for( int *p = v.begin + 1; p != v.end; ++
                        p )
91                  {
92                      v1.end = v2.begin = p;
93                      cost_aux = min_cost( v1 ) + min_cost(
                            v2 );
94                      if( cost_aux < cost_value )
95                          cost_value = cost_aux;
96                  }
97                  cost_value += cost_base;
98              }
99          }
100     }
101     return cost_value;
102 }
103
104
105 int main ()
106 {
107     int length, cuts;
108
109     while( cin >> length, length )
110     {
111         reset( length );
112         cin >> cuts;
113         for( int i = 0; i < cuts; ++i )
114         {
115             cin >> *++vec.end;
116         }
117         *++vec.end = length;
118         cout << "The minimum cutting is " << min_cost(
                vec ) << ".\n";
119     }
120
121     return 0;
122 }
```

A.34. Flowers Flourish from France

```
1  #include <cstdio>
2  #include <cctype>
3
4  #define LAST '*'
5  #define SPACE '␣'
6  #define NEWLINE '\n'
7
8
9  inline char next_letter ()
10 {
11     return toupper (getchar ());
12 }
13
14 inline char next_space ()
15 {
16     char c;
17     do
18     {
19         c = getchar ();
20     }
21     while (!isspace (c));
22     return c;
23 }
24
25
26 inline bool search_tautogram (bool &found)
27 {
28     char t, c;
29
30     t = next_letter ();
31     if (t == LAST)
32         return false;
33
34     found = true;
35     c = next_space ();
36     while (c != NEWLINE && found)
37     {
38         c = next_letter ();
39         if (c != t)
40             found = false;
41         c = next_space ();
42     }
43     while (c != NEWLINE)
```

```
44          c = next_space ();
45
46      return true;
47  }
48
49
50  int main ()
51  {
52      bool tautogram;
53
54      while (search_tautogram (tautogram))
55      {
56          putchar (tautogram ? 'Y' : 'N');
57          putchar ('\n');
58      }
59
60      return 0;
61  }
```

Apéndice B

Algoritmos no aceptados por error al enviar

B.1. Maximum Square

```
1  #include <iostream>
2  using namespace std;
3
4  #define MAX_R 1000
5  #define MAX_C 1000
6  #define DIRTY -1
7
8  int ms, msc[MAX_R][MAX_C], m[MAX_R][MAX_C];
9
10 inline void reset( int R, int C )
11 {
12     ms = 0;
13     for (int i = 0; i < R; ++i)
14     {
15         for (int j = 0; j < C; ++j)
16         {
17             msc[i][j] = DIRTY;
18         }
19     }
20 }
21
22 int maximum_square_corner( int r, int c )
23 {
24     int &result = msc[r][c], r_aux;
25
26     if( result == DIRTY )
27     {
28         r_aux = r * c;
29         if( r > 0 )
30         {
31             r_aux = min( r_aux, maximum_square_corner( r
                   - 1, c ) );
32             if( c > 0 )
33             {
34                 r_aux = min( r_aux, maximum_square_corner
                       ( r - 1, c - 1 ) );
35             }
36         }
37         if( c > 0 )
38         {
39             r_aux = min( r_aux, maximum_square_corner( r,
                       c - 1 ) );
40         }
```

```
41          if( r == 0 && c == 0 )
42          {
43              r_aux = 0;
44          }
45          ++r_aux;
46          result = m[r][c] ? r_aux : 0;
47          ms = max( ms, result );
48
49      }
50
51      return result;
52  }
53
54
55  inline int maximum_square( int R, int C )
56  {
57      maximum_square_corner( R - 1, C - 1 );
58      return ms;
59  }
60
61
62  int main ()
63  {
64      int R, C, r, c;
65
66      while( cin >> R >> C, R || C )
67      {
68          reset( R, C );
69          for( r = 0; r < R; ++r )
70          {
71              for( c = 0; c < C; ++c )
72              {
73                  cin >> m[r][c];
74              }
75          }
76          cout << maximum_square( R, C ) << endl;
77      }
78
79      return 0;
80  }
```