Temas varios

VI Campamento Caribeño de Entrenamiento para el ACM-ICPC

Fidel I. Schaposnik (UNLP) - fidel.s@gmail.com

23 de marzo de 2015

Contenidos

- Algo más de matemática
  - Fábrica de primos
  - FFT
- Geometría computacional
  - Apología de la geometría
  - Representación de objetos fundamentales
    - Puntos y vectores
    - Líneas y segmentos
  - Problemas lineales: intersección de planos en 3D
  - Técnicas de barrido: *November Rain*
  - Par de puntos más cercanos en $\mathcal{O}(N)$
- Steiner tree problem

Fábrica de primos

¿Existe un punto medio en la disyuntiva tiempo vs. memoria en los algoritmos para generar números primos?

Fábrica de primos

¿Existe un punto medio en la disyuntiva tiempo vs. memoria en los algoritmos para generar números primos?

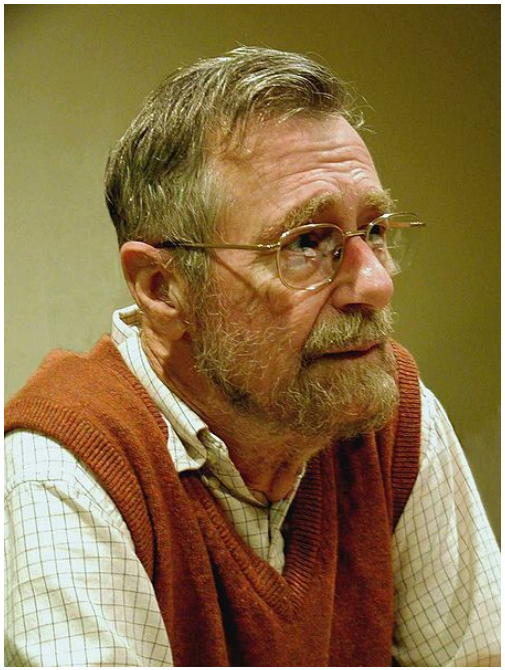


Figure: Edsger W. Dijkstra (1930 - 2002)

E. W. Dijkstra Archive - http://www.cs.utexas.edu/∼EWD/

Fábrica de primos

Dos alternativas aparentemente excluyentes:

- La criba de Eratóstenes opera con un solo primo a la vez, pero trabaja con todos los números que tenemos para procesar

  poco tiempo vs. mucha memoria

- El algoritmo ingenuo opera con todos los primos consecutivamente sobre cada uno de los números que tenemos para procesar

  mucho tiempo vs. poca memoria

Fábrica de primos

Dos alternativas aparentemente excluyentes:

- La criba de Eratóstenes opera con un solo primo a la vez, pero trabaja con todos los números que tenemos para procesar

  poco tiempo vs. mucha memoria

- El algoritmo ingenuo opera con todos los primos consecutivamente sobre cada uno de los números que tenemos para procesar

  mucho tiempo vs. poca memoria

La "línea de montar" trabaja con cada primo por separado pero concurrentemente, operando sobre los números que llegan de a uno

Algoritmo de línea de montaje

- Mantenemos un conjunto de primos ya descubiertos, a cada uno de los cuales le corresponde un menor múltiplo no descartado aún;
- Procesamos los números uno por uno:
  - Si el menor múltiplo de un primo aún no descartado es igual al número considerado, entonces este es compuesto y pasamos al siguiente;
  - Si es mayor, entonces es un nuevo número primo y lo agregamos a nuestro conjunto.

Algoritmo de línea de montaje

- Mantenemos un conjunto de primos ya descubiertos, a cada uno de los cuales le corresponde un menor múltiplo no descartado aún;
- Procesamos los números uno por uno:
  - Si el menor múltiplo de un primo aún no descartado es igual al número considerado, entonces este es compuesto y pasamos al siguiente;
  - Si es mayor, entonces es un nuevo número primo y lo agregamos a nuestro conjunto.

Podemos combinar las optimizaciones usuales para los dos algoritmos conocidos:

- Consideramos solamente números primos hasta $\sqrt{N}$
- Comenzamos a tachar múltiplos a partir de p^2
- Podemos incorporar una rueda

Algoritmo de línea de montaje (código)

```
1  heap.insert(make_pair(4,2)); p[P++] = 2;
2  for (i=3; i<MAXN; i++) {
3    it = heap.lower_bound(make_pair(i,0));
4    if (it->first > i) {
5      p[P++] = i;
6      if (i*i < MAXN) heap.insert(make_pair(i*i,2LL*i));
7    } else {
8      do {
9        heap.insert(make_pair(it->first+it->second,it->second));
10       heap.erase(it);
11       it = heap.lower_bound(make_pair(i,0));
12     } while (it->first == i);
13   }
14 }
```

Algoritmo de la línea de montaje para hallar números primos

El tiempo de ejecución es $\mathcal{O}(N \log N)$ por el uso del heap, y requerimos solamente $\mathcal{O}(\sqrt{N})$ memoria.

Transformada de Fourier

La transformada de Fourier de una función $f(x)$ se define como

$$\tilde{f}(\omega) \equiv \mathcal{F}[f(x)] = \int_{-\infty}^{\infty} f(x) e^{-2\pi i x \omega} dx$$

Bajo ciertas condiciones podemos recuperar la función original a partir de su transformada mediante la anti-transformada

$$\mathcal{F}^{-1}[\tilde{f}(\omega)] \equiv f(x) = \int_{-\infty}^{\infty} \tilde{f}(\omega) e^{2\pi i \omega x} d\omega$$

Entre otras propiedades muy importantes (e.g. la resumación de Poisson $\sum_n \tilde{f}(n) = \sum_n f(n)$), destacamos que para la convolución

$$(f \star g)(x) = \int_{-\infty}^{\infty} f(y) g(x-y) dy \quad \Longrightarrow \quad \mathcal{F}[(f \star g)(x)] = \tilde{f}(\omega) \cdot \tilde{g}(\omega)$$

Transformada de Fourier discreta

Podemos definir una transformada de Fourier para secuencias discretizando los espacios de coordenadas y frecuencias

$$\mathcal{F}_D[f_n] \equiv \tilde{f}_k = \sum_{n=0}^{N-1} f_n e^{-2\pi i k n/N}$$

$$(f \star g)_n = \sum_{m=0}^{N-1} f_m \cdot g_{(n-m) \bmod N} \quad \Longrightarrow \quad \mathcal{F}_D[(f \star g)_n] = \tilde{f}_k \cdot \tilde{g}_k$$

Transformada de Fourier discreta

Podemos definir una transformada de Fourier para secuencias discretizando los espacios de coordenadas y frecuencias

$$\mathcal{F}_D[f_n] \equiv \tilde{f}_k = \sum_{n=0}^{N-1} f_n e^{-2\pi i k n/N}$$

$$(f \star g)_n = \sum_{m=0}^{N-1} f_m \cdot g_{(n-m) \bmod N} \quad \Longrightarrow \quad \mathcal{F}_D[(f \star g)_n] = \tilde{f}_k \cdot \tilde{g}_k$$

- A la izquierda, calcular la información contenida en la convolución toma $\mathcal{O}(N^2)$
- A la derecha, la misma información puede calcularse en $\mathcal{O}(N)$

Si podemos transformar y anti-transformar en tiempo asintótico mejor que $\mathcal{O}(N^2)$, será conveniente hacer

$$\{f_n, g_n\} \Longrightarrow \mathcal{F}_D \Longrightarrow \{\tilde{f}_k, \tilde{g}_k\} \Longrightarrow f_k \cdot g_k \Longrightarrow \mathcal{F}_D^{-1} \Longrightarrow (f \star g)_n$$

FFT

Consideremos por simplicidad N par, y observemos que

$$\tilde{f}_k = \sum_{n=0}^{N-1} f_n e^{-\frac{2\pi i}{N} kn} = \sum_{m=0}^{N/2-1} f_{2m} e^{-\frac{2\pi i}{N/2} km} + e^{-\frac{2\pi i}{N} k} \sum_{m=0}^{N/2-1} f_{2m+1} e^{-\frac{2\pi i}{N/2} mk}$$

Si llamamos a_n a la subsecuencia de elementos pares de f_n, y b_n a la subsecuencia de sus elementos impares, tenemos

$$\tilde{f}_k = \tilde{a}_k + e^{-\frac{2\pi i}{N} k} \tilde{b}_k$$

- $\tilde{f}_k$ tiene periodo N
- $\tilde{a}_k$ y $\tilde{b}_k$ tienen periodo $N/2$
- El factor de twiddle $e^{-\frac{2\pi i}{N} k}$ es anti-periódico con $k \mapsto k + \frac{N}{2}$

FFT (cont.)

Tenemos entonces

$$\tilde{f}_k = \begin{cases} \tilde{f}_k = \tilde{a}_k + e^{-\frac{2\pi i}{N}k}\tilde{b}_k \\ \tilde{f}_{k+\frac{N}{2}} = \tilde{a}_k - e^{-\frac{2\pi i}{N}k}\tilde{b}_k \end{cases} \qquad \text{para} \qquad k = 0, \ldots, N/2$$

FFT (cont.)

Tenemos entonces

$$\tilde{f}_k = \begin{cases} \tilde{f}_k = \tilde{a}_k + e^{-\frac{2\pi i}{N}k}\tilde{b}_k \\ \tilde{f}_{k+\frac{N}{2}} = \tilde{a}_k - e^{-\frac{2\pi i}{N}k}\tilde{b}_k \end{cases} \qquad \text{para} \qquad k = 0, \ldots, N/2$$

¡Reducimos el tamaño del problema a la mitad y recombinamos los resultados en $\mathcal{O}(N)$!

FFT (cont.)

Tenemos entonces

$$\tilde{f}_k = \begin{cases} \tilde{f}_k = \tilde{a}_k + e^{-\frac{2\pi i}{N}k}\tilde{b}_k \\ \tilde{f}_{k+\frac{N}{2}} = \tilde{a}_k - e^{-\frac{2\pi i}{N}k}\tilde{b}_k \end{cases} \qquad \text{para} \qquad k = 0, \ldots, N/2$$

¡Reducimos el tamaño del problema a la mitad y recombinamos los resultados en $\mathcal{O}(N)$!

- El tiempo de ejecución es $\mathcal{O}(N \log N)$
- La elección del punto de división no es ingenua: siempre debemo utilizar un divisor de N
- En general podemos elegir la cantidad de "puntos de sampleo" para tener $N = 2^k$, o extender con 0's una secuencia finita hasta la siguiente potencia de dos.

FFT (código)

```
1  typedef complex<double> cpx;
2  const double dos_pi = 4.0*acos(0.0);
3
4  void fft(cpx x[], cpx y[], int dx, int N, int dir) {
5    if (N > 1) {
6      fft(x, y, 2*dx, N/2, dir);
7      fft(x+dx, y+N/2, 2*dx, N/2, dir);
8      for (int i=0; i<N/2; i++) {
9        cpx even = y[i], odd = y[i+N/2],
             twiddle=exp(cpx(0,dir*dos_pi*i/N));
10       y[i] = even+twiddle*odd;
11       y[i+N/2] = even-twiddle*odd;
12     }
13   } else y[0] = x[0];
14 }
```

FFT con el algoritmo de CooleyTukey para $N = 2^k$

- $\tilde{f} = \mathcal{F}_D[f] \quad \Longrightarrow \quad fft(f, \tilde{f}, 1, N, 1)$
- $f = \mathcal{F}_D^{-1}[\tilde{f}] \quad \Longrightarrow \quad fft(\tilde{f}, f, 1, N, -1)$

Geometría computacional

¿En qué consiste un problema de geometría computacional?

Generalmente, los problemas de geometría en el ICPC requieren calcular alguna cantidad (e.g. distancia, área, parámetros óptimos para una configuración determinada, etc) relacionada con elementos geométricos como puntos, líneas, círculos, y demás. La solución de un problema de geometría involucra dos pasos:

1. Representar los objetos geométricos involucrados en el problema, de modo de poder operar con ellos.
2. Desarrollar un algoritmo utilizando lo anterior para calcular la respuesta buscada.

Para poder dedicarnos a la parte "interesante", debemos primero asegurarnos de que el primer punto no es un impedimento.

Apología de la geometría

A FAVOR: Podemos *visualizar* los algoritmos fácilmente.

EN CONTRA: Debemos evitar nuevos *tipos* de problemas (errores numéricos, casos degenerados, código más largo, etc).

Apología de la geometría

A FAVOR: Podemos *visualizar* los algoritmos fácilmente.

EN CONTRA: Debemos evitar nuevos *tipos* de problemas (errores numéricos, casos degenerados, código más largo, etc).

Antes de empezar, algunas consideraciones generales

- A lo largo de esta clase nos vamos a concentrar casi exclusivamente en problemas de geometría en 2D.
- La **práctica** y las **buenas costumbres** son todavía más importantes que en los problemas "convencionales".
- ¡Los problemas de geometría muchas veces sirven para distinguir a los buenos equipos de los excelentes!

Puntos y vectores

Un punto en el plano (o un vector desde el origen hasta dicho punto) se puede representar con un par ordenado de coordenadas en un sistema cartesiano:

$$\vec{P} = (x, y) \qquad \text{con} \qquad x, y \in \mathbb{R}$$

Algunas operaciones entre vectores

- La suma y resta de vectores se realiza componente a componente:

$$\vec{P} = \vec{P}_1 \pm \vec{P}_2 \qquad \Longleftrightarrow \qquad (x, y) = (x_1 \pm x_2, y_1 \pm y_2)$$

- La longitud o *norma* de un vector es $|\vec{P}| = \sqrt{x^2 + y^2}$.
- La distancia euclídea entre dos puntos $\vec{P}_1$ y $\vec{P}_2$ es $|\vec{P}_1 - \vec{P}_2|$
- El producto escalar de dos vectores es un número, y se define como

$$\vec{P}_1 \cdot \vec{P}_2 := x_1 x_2 + y_1 y_2 = |\vec{P}_1||\vec{P}_2| \cos\theta$$

Observar que es la proyección de un vector sobre otro, y en particular si $\theta = 90°$ el producto escalar se anula.

Puntos y vectores (cont.)

- El producto vectorial de dos vectores es un vector en la dirección normal al plano generado por ellos. En dos dimensiones nos puede interesar su única componente no nula,

$$\left(\vec{P}_1 \times \vec{P}_2\right)_z = x_1 y_2 - x_2 y_1 \equiv \vec{P}_1 \wedge \vec{P}_2$$

Observar que

$$|\vec{P}_1 \times \vec{P}_2| = |\vec{P}_1||\vec{P}_2| \sin\theta,$$

es decir el área del paralelogramo formado por los vectores y sus traslaciones paralelas.

En particular si $\theta = 0°$ ó $\theta = 180°$ el producto vectorial se anula.

Puntos y vectores (código.)

```
1  struct pt {
2      double x, y;
3      pt(double xx=0.0, double yy=0.0) { x=xx; y=yy; }
4  };
5
6  pt operator+(const pt &p1, const pt &p2) {
7      return pt(p1.x+p2.x, p1.y+p2.y); }
8
9  pt operator-(const pt &p1, const pt &p2) {
10     return pt(p1.x-p2.x, p1.y-p2.y); }
11
12 double operator*(const pt &p1, const pt &p2) {
13     return p1.x*p2.x + p1.y*p2.y; }
14
15 double operator^(const pt &p1, const pt &p2) {
16     return p1.x*p2.y - p1.y*p2.x; }
17
18 double norm(const pt &p) { return sqrt(p*p); }
19
20 double dist(const pt &p1, const pt &p2) {
21     return norm(p1-p2); }
```

Estructura y operaciones elementales con puntos

Líneas y segmentos

Una línea o un segmento se pueden representar de varias formas distintas:

- Con dos puntos $\vec{P}_1$ y $\vec{P}_2$ sobre la línea (los extremos del segmento)
- Con un punto $\vec{P}_0$ sobre la línea y un vector director $\vec{V}$
- Mediante una ecuación implícita $ax + by + c = 0$.

Se puede pasar de una a otra representación trivialmente, y la elección de una u otra depende en definitiva del uso que se vaya a darle.

La representación $\left\{\vec{P}_0, \vec{V}\right\}$ se conoce como paramétrica, porque nos permite recorrer todos los puntos de la línea o segmento con

$$\vec{P}(t) = \vec{P}_0 + t\vec{V} \qquad \text{con} \qquad \begin{cases} t \in \mathbb{R} & \text{línea} \\ t \in [0,1] & \text{segmento} \end{cases}$$

Líneas y segmentos (cont.)

```
1  struct line {
2      double a, b, c;
3      line(double aa=0.0, double bb=0.0, double cc=0.0) {
4          a=aa; b=bb; c=cc;
5      }
6  };
7
8  double dist(const pt &p, const line &l) {
9      return ABS(l.a*p.x+l.b*p.y+l.c)/sqrt(SQ(l.a)+SQ(l.b)); }
10
11 line line_pp(const pt &p1, const pt &p2) {
12     return line(p2.y-p1.y, p1.x-p2.x, p2^p1); }
13
14 line line_perp_p(const line &l, const pt &p) {
15     return line(-l.b, l.a, l.b*p.x - l.a*p.y); }
16
17 line mediatriz(const pt &p1, const pt &p2) {
18     return line_perp_p(line_pp(p1, p2), (p1+p2)/2.0); }
```

Ejemplo de uso de la representación implícita

Líneas y segmentos (cont.)

```
1  line bisectriz(const pt &p1, const pt &pc, const pt &p2) {
2      pt pc1 = p1-pc, pc2 = p2-pc;
3
4      if (ABS(pc1^pc2) < EPS) {
5          if (pc1*pc2 > ZERO) return line_pp(pc, p1);
6          else return line_perp_p(line_pp(p1, p2), pc);
7      }
8      return line_pp(pc,pc+(pc1/norm(pc1)+pc2/norm(pc2))/2.0);
9  }
10
11 int inter_ll(const line &l1, const line &l2, pt &p) {
12     double det = l1.a*l2.b - l1.b*l2.a;
13
14     if (ABS(det) < EPS) {
15         if (ABS(l1.a*l2.c - l1.c*l2.a) < EPS) return -1;
16         else return 0;
17     }
18     p.x = (l1.b*l2.c - l2.b*l1.c)/det;
19     p.y = (l2.a*l1.c - l1.a*l2.c)/det;
20     return 1;
21 }
```

Ejemplo de uso de la representación implícita (cont.)

Problemas lineales en geometría computacional

Consideremos la intersección de dos planos en tres dimensiones.

Problemas lineales en geometría computacional

Consideremos la intersección de dos planos en tres dimensiones.

Un plano se define con un punto $\vec{P_0}$ contenido en el plano y un vector $\vec{N_0}$ ortogonal al mismo. Todos los otros puntos satisfacen

$$(\vec{P} - \vec{P_0}) \cdot \vec{N_0} = 0$$

lo cual conduce a la ecuación más familiar

$$ax + by + cz + d = 0 \quad \text{para} \quad \vec{N_0} = (a, b, c) \quad \text{y} \quad d = -\vec{P_0} \cdot \vec{N_0}$$

En general, dos planos definidos por $\{\vec{P_1}, \vec{N_1}\}$ y $\{\vec{P_2}, \vec{N_2}\}$ se intersecan para dar una línea. **¿Cómo la hallamos?**

Problemas lineales en geometría computacional (cont.)

Usamos la representación paramétrica de la línea, de modo que buscamos un punto $\vec{P}$ sobre ella y un vector director $\vec{V}$.

Problemas lineales en geometría computacional (cont.)

Usamos la representación paramétrica de la línea, de modo que buscamos un punto $\vec{P}$ sobre ella y un vector director $\vec{V}$.

Nuestra línea debe pertenecer a ambos planos, luego es ortogonal a los vectores $\vec{N}_1$ y $\vec{N}_2$. Entonces

$$\vec{V} = \vec{N}_1 \times \vec{N}_2$$

Si $\vec{V} = \vec{0}$, los planos son paralelos y no hay intersección entre ellos.
¿Cómo hallamos $\vec{P}$?

Problemas lineales en geometría computacional (cont.)

Usamos la representación paramétrica de la línea, de modo que buscamos un punto $\vec{P}$ sobre ella y un vector director $\vec{V}$.

Nuestra línea debe pertenecer a ambos planos, luego es ortogonal a los vectores $\vec{N}_1$ y $\vec{N}_2$. Entonces

$$\vec{V} = \vec{N}_1 \times \vec{N}_2$$

Si $\vec{V} = \vec{0}$, los planos son paralelos y no hay intersección entre ellos.
¿Cómo hallamos $\vec{P}$?

Sea $\vec{P}$ el punto sobre la línea que está más cerca de cierto punto fijo y arbitrario (*e.g.* el origen). Entonces $\vec{P} = (x, y, z)$ minimiza

$$D^2 = x^2 + y^2 + z^2$$

mientras está en ambos planos, *i.e.* satisfaciendo

$$\left(\vec{P} - \vec{P}_1\right) \cdot \vec{N}_1 = 0 \quad \text{y} \quad \left(\vec{P} - \vec{P}_2\right) \cdot \vec{N}_2 = 0$$

Problemas lineales en geometría computacional (cont.)

Podemos hallar el punto $\vec{P}$ minimizando D^2 bajo estas restricciones, usando multiplicadores de Lagrange. Definimos

$$f(x, y, z, \mu_1, \mu_2) = x^2+y^2+z^2+\mu_1 \left(\vec{P} - \vec{P}_1\right)\cdot\vec{N}_1+\mu_2 \left(\vec{P} - \vec{P}_2\right)\cdot\vec{N}_2$$

Luego debemos exigir

$$\frac{\partial f}{\partial x} = 2x+\mu_1 x_{N_1} = 0 \qquad \frac{\partial f}{\partial y} = 2y+\mu_1 y_{N_1} = 0 \qquad \frac{\partial f}{\partial z} = 2z+\mu_1 z_{N_1} = 0$$

$$\frac{\partial f}{\partial \mu_1} = \left(\vec{P} - \vec{P}_1\right)\cdot\vec{N}_1 = 0 \qquad \frac{\partial f}{\partial \mu_2} = \left(\vec{P} - \vec{P}_2\right)\cdot\vec{N}_2 = 0$$

Problemas lineales en geometría computacional (cont.)

Podemos hallar el punto $\vec{P}$ minimizando D^2 bajo estas restricciones, usando multiplicadores de Lagrange. Definimos

$$f(x, y, z, \mu_1, \mu_2) = x^2+y^2+z^2+\mu_1 \left(\vec{P} - \vec{P}_1\right)\cdot\vec{N}_1+\mu_2 \left(\vec{P} - \vec{P}_2\right)\cdot\vec{N}_2$$

Luego debemos exigir

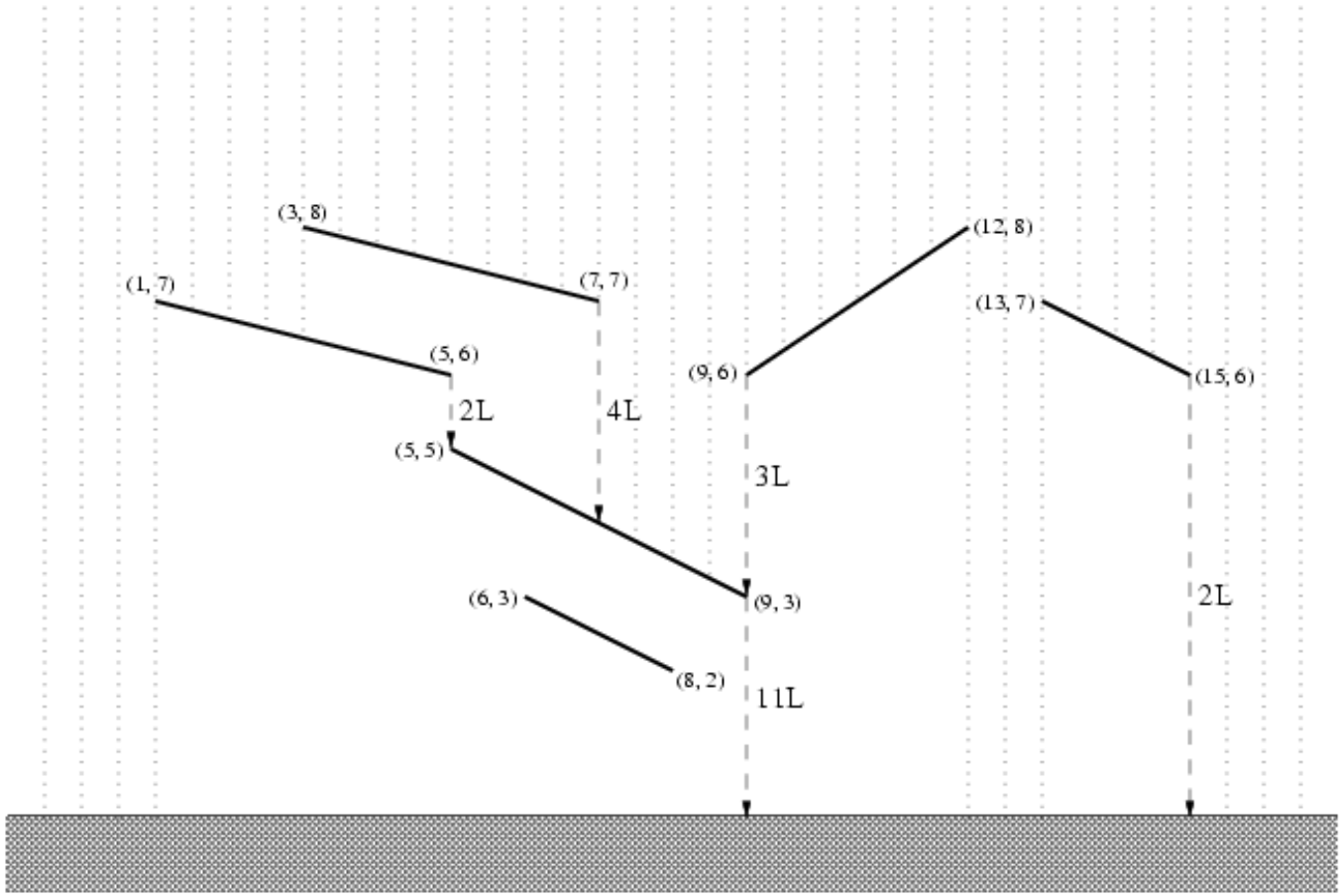
$$\frac{\partial f}{\partial x} = 2x+\mu_1 x_{N_1} = 0 \qquad \frac{\partial f}{\partial y} = 2y+\mu_1 y_{N_1} = 0 \qquad \frac{\partial f}{\partial z} = 2z+\mu_1 z_{N_1} = 0$$

$$\frac{\partial f}{\partial \mu_1} = \left(\vec{P} - \vec{P}_1\right)\cdot\vec{N}_1 = 0 \qquad \frac{\partial f}{\partial \mu_2} = \left(\vec{P} - \vec{P}_2\right)\cdot\vec{N}_2 = 0$$

O, en notación matricial

$$\begin{pmatrix} 2 & 0 & 0 & x_{N_1} & x_{N_2} \\ 0 & 2 & 0 & y_{N_1} & y_{N_2} \\ 0 & 0 & 2 & z_{N_1} & z_{N_2} \\ x_{N_1} & y_{N_1} & z_{N_1} & 0 & 0 \\ x_{N_2} & y_{N_2} & z_{N_2} & 0 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ \mu_1 \\ \mu_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ \vec{P}_1\cdot\vec{N}_1 \\ \vec{P}_2\cdot\vec{N}_2 \end{pmatrix}$$

Técnica de barrido: *November Rain* (CERC/SWERC 2003)



¿Cantidad de agua recolectada por cada segmento de techo?

November Rain (solución)

Representamos a los segmentos de techo como los nodos de un grafo

- Un nodo especial S (¿las nubes?) actúa como fuente de toda el agua
- Tenemos una arista entre dos nodos si el agua cae de uno de ellos al otro
- La gravedad se encarga de que el grafo sea un DAG (nunca llueve hacia arriba)

November Rain (solución)

Representamos a los segmentos de techo como los nodos de un grafo

- Un nodo especial S (¿las nubes?) actúa como fuente de toda el agua
- Tenemos una arista entre dos nodos si el agua cae de uno de ellos al otro
- La gravedad se encarga de que el grafo sea un DAG (nunca llueve hacia arriba)

Tenemos dos problemas de geometría computacional antes de reducir el problema a uno de grafos

1. Identificar las aristas presentes
2. Calcular los pesos de las aristas que salen de la fuente

November Rain (cont. solución)

Imaginemos una línea vertical que se mueve de izquierda a derecha (dirección x creciente). A medida que avanza pueden ocurrir tres tipos de eventos:

1. Comienza a intersecar el segmento de techo i (en $x = x_1[i]$)
2. Termina de intersecar el segmento de techo i (en $x = x_2[i]$)
3. Coincide momentáneamente con un segmento vertical por donde cae el agua recolectada por el techo i (en $x = x_1[i]$ si $y_1[i] < y_2[i]$, o en $x = x_2[i]$ si $y_1[i] > y_2[i]$)

November Rain (cont. solución)

Imaginemos una línea vertical que se mueve de izquierda a derecha (dirección x creciente). A medida que avanza pueden ocurrir tres tipos de eventos:

1. Comienza a intersecar el segmento de techo i (en $x = x_1[i]$)
2. Termina de intersecar el segmento de techo i (en $x = x_2[i]$)
3. Coincide momentáneamente con un segmento vertical por donde cae el agua recolectada por el techo i (en $x = x_1[i]$ si $y_1[i] < y_2[i]$, o en $x = x_2[i]$ si $y_1[i] > y_2[i]$)

Con estos eventos podemos mantener un conjunto de segmentos de techo "activos" (intersecados por la línea de barrido) y

1. Sumar al peso de la arista $S \to i$ el la longitud que avanza el barrido siendo i el techo activo más alto
2. Calcular el primer nodo activo debajo del techo i cuando ocurre un evento de tipo 3

Par de puntos más cercanos en $\mathcal{O}(N)$

Dado un conjunto S de N puntos en el plano

$$S = \{\vec{p}_1, \ldots, \vec{p}_N\}$$

queremos hallar la distancia $\delta(S)$ entre el par de puntos más cercanos en S.

- Existe una cota inferior de $\Omega(N \log N)$ para este problema (para determinar si hay dos elementos distintos en S, i.e. responder ¿$\delta(S) = 0$?);

Par de puntos más cercanos en $\mathcal{O}(N)$

Dado un conjunto S de N puntos en el plano

$$S = \{\vec{p}_1, \ldots, \vec{p}_N\}$$

queremos hallar la distancia $\delta(S)$ entre el par de puntos más cercanos en S.

- Existe una cota inferior de $\Omega(N \log N)$ para este problema (para determinar si hay dos elementos distintos en S, i.e. responder ¿$\delta(S) = 0$?);
- Vamos a dar un algoritmo de tiempo esperado $\mathcal{O}(N)$

Par de puntos más cercanos en $\mathcal{O}(N)$

Dado un conjunto S de N puntos en el plano

$$S = \{\vec{p}_1, \ldots, \vec{p}_N\}$$

queremos hallar la distancia $\delta(S)$ entre el par de puntos más cercanos en S.

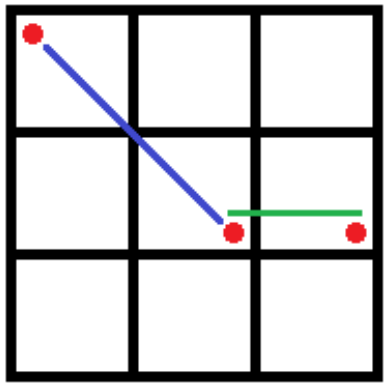
- Existe una cota inferior de $\Omega(N \log N)$ para este problema (para determinar si hay dos elementos distintos en S, i.e. responder ¿$\delta(S) = 0$?);
- Vamos a dar un algoritmo de tiempo esperado $\mathcal{O}(N)$

El algoritmo utiliza $\lfloor \cdot \rfloor$ y por lo tanto la cota no se aplica =)

Par de puntos más cercanos en $\mathcal{O}(N)$ (cont.)

- Utilizamos una "criba" para filtrar puntos de S en pasos sucesivos, eliminando aquellos puntos que están aislados (demasiado lejos de todos los demás puntos de nuestro conjunto).
- El filtrado no será perfecto, pero podemos corregir nuestro resultado en un paso final.

Para filtrar utilizamos una grilla:



Asignamos el punto (x, y) a la celda $(\lfloor \frac{x}{L} \rfloor, \lfloor \frac{y}{L} \rfloor)$

Par de puntos más cercanos en $\mathcal{O}(N)$ (cont.)

Sea $N(\vec{p})$ el vecindario del punto $\vec{p}$:

- Si $|\vec{p} - \vec{p}'| < L \Longrightarrow \vec{p}' \in N(\vec{p})$
- Si $|\vec{p} - \vec{p}'| > 2\sqrt{2}L \Longrightarrow \vec{p}' \notin N(\vec{p})$
- Si $L \leq |\vec{p} - \vec{p}'| \leq 2\sqrt{2}L \Longrightarrow$ puede darse cualquiera de las dos

Usamos un filtro F para definir una secuencia de conjuntos $S_0, S_1, \ldots,$ siendo

$$S_0 = S \qquad \text{y} \qquad S_{i+1} = F(S_i)$$

Dado un conjunto S_i, el filtro

1. Toma un punto al azar $\vec{p}_i^* \in S_i$
2. Calcula la mínima distancia $d_i(\vec{p}_i^*)$ de $\vec{p}_i^*$ a cualquier otro punto de S_i
3. Crea una grilla de lado $L_i = d_i(\vec{p}_i^*)/3$
4. Descarta todo punto $\vec{p} \in S_i$ tal que $N(\vec{p}) = \{\vec{p}\}$ para dar S_{i+1}

Par de puntos más cercanos en $\mathcal{O}(N)$ (cont.)

Dadas las propiedades de la grilla, sabemos que:

- Todo $\vec{p}$ tal que $d_i(\vec{p}) > 3L_i = d_i(\vec{p}_i^*)$ fue descartado
- Todo $\vec{p}$ tal que $d_i(\vec{p}) < L_i = d_i(\vec{p}_i^*)/3$ no fue descartado

Par de puntos más cercanos en $\mathcal{O}(N)$ (cont.)

Dadas las propiedades de la grilla, sabemos que:

- Todo $\vec{p}$ tal que $d_i(\vec{p}) > 3L_i = d_i(\vec{p}_i^*)$ fue descartado
- Todo $\vec{p}$ tal que $d_i(\vec{p}) < L_i = d_i(\vec{p}_i^*)/3$ no fue descartado

Por lo tanto:

- $d_i(\vec{p}_i^*)$ es una secuencia decreciente
- El proceso termina en una cantidad finita K de pasos
- $d_K(\vec{p}_K^*)$ es una aproximación a $\delta(S)$
- Tenemos la cota

$$\frac{d_K(\vec{p}_K^*)}{3} \leq \delta(S) \leq d_K(\vec{p}_K^*)$$

Par de puntos más cercanos en $\mathcal{O}(N)$ (cont.)

Para hallar el resultado exacto, creamos una grilla de lado $d_K(\vec{p}^*)$ y sabemos que

$$|\vec{p} - \vec{p}'| \leq d_K(\vec{p}_K^*) \qquad \Longrightarrow \qquad \vec{p}' \in N(\vec{p})$$

Para cada $\vec{p} \in S$ calculamos $\delta(N(\vec{p}))$ por fuerza bruta porque $|N(\vec{p})| = \mathcal{O}(1)$.

Par de puntos más cercanos en $\mathcal{O}(N)$ (cont.)

Para hallar el resultado exacto, creamos una grilla de lado $d_K(\vec{p}^*)$ y sabemos que

$$|\vec{p} - \vec{p}'| \leq d_K(\vec{p}_K^*) \qquad \Longrightarrow \qquad \vec{p}' \in N(\vec{p})$$

Para cada $\vec{p} \in S$ calculamos $\delta(N(\vec{p}))$ por fuerza bruta porque $|N(\vec{p})| = \mathcal{O}(1)$.

Análisis: En cada paso elegimos un punto $\vec{p}_i^*$ al azar, luego obtenemos un valor $d_i(\vec{p}_i^*)$ al azar en $\{d_i(\vec{p}) : \vec{p} \in S_i\}$. Todos los puntos con $d_i(\vec{p}) > d_i(\vec{p}_i^*)$ se descartan, luego la cantidad esperada de puntos en S_i es

$$E(S_{i+1}) = \frac{E(S_i)}{2} \quad \text{con} \quad E(S_0) = N \quad \Longrightarrow \quad E(S_i) = \frac{N}{2^i}$$

Tiempo esperado $\mathcal{O}\left(\sum_{i=0}^{\log N} E_i\right) = \mathcal{O}(N)$.

Steiner tree problem

Dado un grafo de $|V| = N$ nodos, queremos hallar el árbol de cubrimiento mínimo (en términos de la suma de los pesos de sus aristas) que contiene los K nodos de un subconjunto $U \subseteq V$.

Steiner tree problem

Dado un grafo de $|V| = N$ nodos, queremos hallar el árbol de cubrimiento mínimo (en términos de la suma de los pesos de sus aristas) que contiene los K nodos de un subconjunto $U \subseteq V$.

- Parece similar al problema del MST, sin embargo es NP-hard;
- Es "fixed parameter tractable", es decir que existen algoritmos en tiempo

$$\mathcal{O}(f(K)g(N)) \qquad \text{con} \qquad f(x) \sim \exp \quad \text{pero} \quad g(x) \sim \text{pol}$$

Steiner tree problem

Dado un grafo de $|V| = N$ nodos, queremos hallar el árbol de cubrimiento mínimo (en términos de la suma de los pesos de sus aristas) que contiene los K nodos de un subconjunto $U \subseteq V$.

- Parece similar al problema del MST, sin embargo es NP-hard;
- Es "fixed parameter tractable", es decir que existen algoritmos en tiempo

$$\mathcal{O}(f(K)g(N)) \qquad \text{con} \qquad f(x) \sim \exp \quad \text{pero} \quad g(x) \sim \text{pol}$$

Solución intuida: ¿programación dinámica con subconjuntos $W \subseteq U$?

Steiner tree problem

Dado un grafo de $|V| = N$ nodos, queremos hallar el árbol de cubrimiento mínimo (en términos de la suma de los pesos de sus aristas) que contiene los K nodos de un subconjunto $U \subseteq V$.

- Parece similar al problema del MST, sin embargo es NP-hard;
- Es "fixed parameter tractable", es decir que existen algoritmos en tiempo

$$\mathcal{O}(f(K)g(N)) \qquad \text{con} \qquad f(x) \sim \exp \quad \text{pero} \quad g(x) \sim \text{pol}$$

Solución intuida: ¿programación dinámica con subconjuntos $W \subseteq U$? Sí, pero con algo más...

Steiner tree problem (solución)

Consideremos un subconjunto $W \subseteq U$ y un nodo adicional $u \in V$. Dado un árbol de cubrimiento mínimo que contenga $W \cup \{u\}$, cuando eliminamos u:

1. O bien separamos a W en dos partes no triviales W_1 y W_2;
2. Caso contrario, u debe estar conectado (por un camino mínimo) con otro nodo $v \in V$ que sí separa a W en dos partes no triviales.

Steiner tree problem (solución)

Consideremos un subconjunto $W \subseteq U$ y un nodo adicional $u \in V$. Dado un árbol de cubrimiento mínimo que contenga $W \cup \{u\}$, cuando eliminamos u:

1. O bien separamos a W en dos partes no triviales W_1 y W_2;
2. Caso contrario, u debe estar conectado (por un camino mínimo) con otro nodo $v \in V$ que sí separa a W en dos partes no triviales.

Luego

- Precalculamos las distancias $d(u, v)$ para todo par de nodos;
- Para cada W, consideramos todas sus particiones posibles en dos subconjuntos disjuntos no triviales W_1 y W_2;
- Consideramos las dos posibilidades de arriba.

Steiner tree problem (código)

```
1 for (i=1; i<(1<<K); i++) {
2   for (j=0; j<N; j++)
3     for (k=(i-1)&i; k; k=(k-1)&i)
4       dp[i][j] = min(dp[i][j], dp[k][j]+dp[i^k][j]);
5   for (j=0; j<N; j++)
6     for (k=0; k<N; k++)
7       dp[i][j] = min(dp[i][j], dp[i][k]+dist[j][k]);
8 }
```

Steiner tree problem en $\mathcal{O}(3^K N + 2^K N^2 + N^3)$

Referencia: Xinhui Wang, "*Exact algorithms for the Steiner tree problem*"

¡Gracias!