Nálægustu punktar í plani

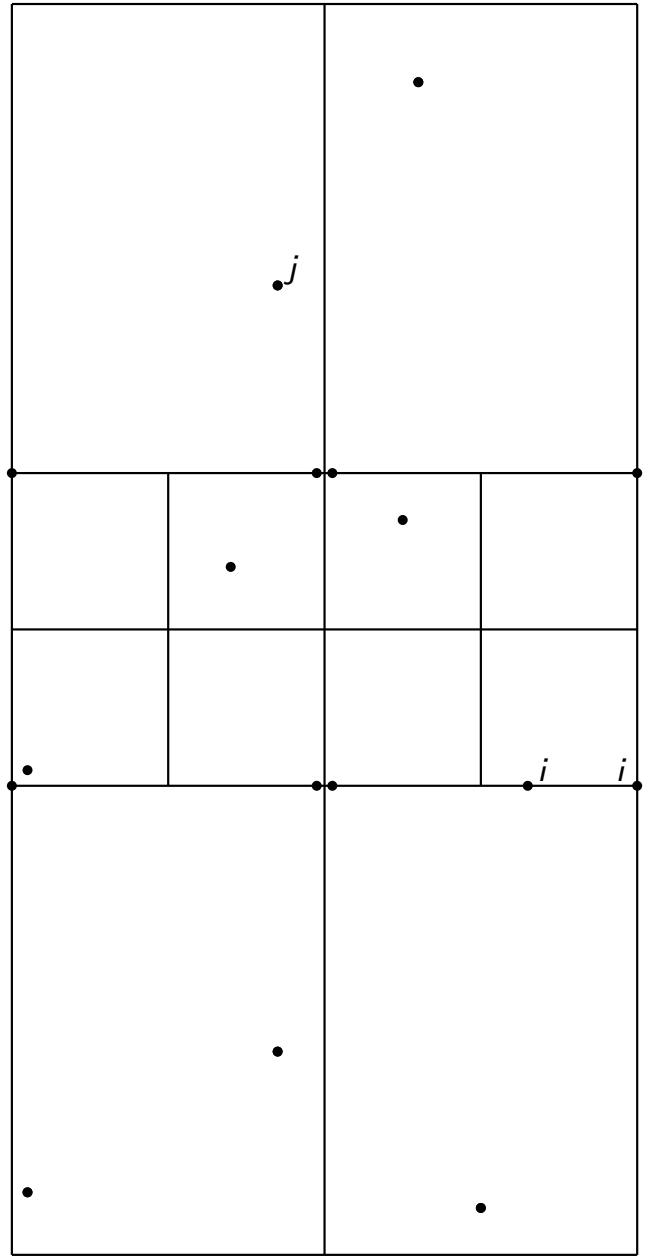
Bergur Snorrason

March 25, 2024

- Gefnir eru n punktar í plani.
- Hvaða tveir punktar hafa minnstu fjarlægð sín á milli?
- Við getum leyst þetta með tæmandi leit í $\mathcal{O}(n^2)$.
- Við skoðum einfaldlega öll pör punkta.
- Prófum að bæta þetta með því að deila og drottna.

- Röðum punktunum eftir x-hniti og skiptum í helminga.
- Látum x_0 vera þannig að hann liggi á milli x-hnita helminganna.
- Leysum svo endurkvæmt fyrir hvorn helming fyrir sig.
- Við þurfum nú að athuga hvort eitthvert par á milli helminganna sé betra en bestu pörin í hvorum helmingnum.
- Það er of hægt að skoða öll pörin sem liggja á milli, þá verður tímaflækjan $\mathcal{O}(n^2)$.
- Látum d vera minnstu fjarlægðina sem fannst í hvorum helmingnum fyrir sig.
- Við getum þá hunsað þá punkta sem hafa x-hnit utan bilsins $[x_0 - d, x_0 + d]$.
- Röðum afgangnum eftir y-hniti.
- Svo kemur trikkið.
- Okkur nægir, fyrir hvern punkt, að skoða fastann fjölda af næstu punktum.

- Skiptum svæðinu fyrir ofan punktinn x_i í átta ferninga, hver með hliðarlengdir $d/2$.
- Ef fjarlægðin milli allra punktana í hvorum helming er ekki minni en d þá getur mest einn punktur verið í hverjum ferningi (þar með talið er x_i).
- Allir punktar utan þessa svæðis eru meira en d fjarlægð frá i-ta punktinum, svo við þurfum ekki að skoða þá.
- Svo við þurfum bara að skoða fjarlægðina frá x_i í x_j þegar $j - i \leq 7$.


j
i
i

```
23 double closest_pair_r(pt *a, pt *b, int n, pt *r1, pt *r2)
24 {
25     if (n < 2) return 1e16;
26     int m = n/2, i, j, k = 0;
27     pt r3, r4;
28     double p = closest_pair_r(a, b, m, r1, r2);
29     double d = closest_pair_r(a + m + 1, b + m + 1, n - m - 1, &r3, &r4);
30     if (d < p) p = d, *r1 = r3, *r2 = r4;
31     for (i = 0; i < n; i++) if (fabs(creal(a[i] - a[m])) < p) b[k++] = a[i];
32     qsort(b, k, sizeof *b, cmpy);
33     for (i = 0; i < k; i++)
34         for (j = i + 1; cimag(b[j] - b[i]) < p && j < k; j++)
35             if (cabs(b[i] - b[j]) < p)
36                 p = cabs(b[i] - b[j]), *r1 = b[i], *r2 = b[j];
37     return p;
38 }
39
40 double closest_pair(pt* a, int n, pt* r1, pt* r2)
41 {
42     pt b[n];
43     qsort(a, n, sizeof *a, cmpx);
44     return closest_pair_r(a, b, n, r1, r2);
45 }
```

- Hvert endurkvæmt kall er $\mathcal{O}(n \log n)$.
- Svo tímaflækjan á þessari útfærslu er $\mathcal{O}(n \log^2 n)$, sem er bæting.
- Eina ástæðan fyrir því að hvert endurkvæmt kall sé $\mathcal{O}(n \log n)$ er að við röðum í hvert skipti.
- Það er óþarfi því við erum alltaf að raða sömu punktunum aftur og aftur.
- Það eru fleiri en ein leið til að þurfa ekki að raða oft.
- Ein leið er að gera það sama og er gert í `mergesort`.
- Þá erum við í raun að raða, en við gerum það í línulegum tíma (í hverju endurkvæma kalli).

```
15 void merge(pt* a, pt *b, int m, int r)
16 {
17     int i = 0, j = m, c = 0;
18     while (i < m && j < r) b[c++] = a[cimag(a[j] - a[i]) < 0.0 ? j++ : i++];
19     while (i < m || j < r) b[c++] = a[j < r ? j++ : i++];
20     for (i = 0; i < r; i++) a[i] = b[i];
21 }
22
23 double closest_pair_r(pt *a, pt *b, pt *c, int n, pt *r1, pt *r2)
24 {
25     if (n < 2) { b[0] = a[0]; return 1e16; }
26     int m = n/2, i, j, k = 0;
27     pt r3, r4;
28     double p = closest_pair_r(a, b, c, m, r1, r2), d;
29     b[m] = a[m];
30     d = closest_pair_r(a + m + 1, b + m + 1, c + m + 1, n - m - 1, &r3, &r4);
31     if (d < p) p = d, *r1 = r3, *r2 = r4;
32     merge(b, c, m, m + 1), merge(b, c, m + 1, n);
33     for (i = 0; i < n; i++) if (fabs(creal(b[i] - a[m])) < p) c[k++] = b[i];
34     for (i = 0; i < k; i++)
35         for (j = i + 1; cimag(c[j] - c[i]) < p && j < k; j++)
36             if (cabs(c[i] - c[j]) < p)
37                 p = cabs(c[i] - c[j]), *r1 = c[i], *r2 = c[j];
38     return p;
39 }
40
41 double closest_pair(pt* a, int n, pt* r1, pt* r2)
42 {
43     pt b[n], c[n];
44     qsort(a, n, sizeof *a, cmpx);
45     return closest_pair_r(a, b, c, n, r1, r2);
46 }
```

```
17 double closest_pair(pt *a, int n, pt *x, pt *y)
18 {
19     double p[n];
20     int i, j, k, l, m, e, r;
21     pt b[n], c[n], z[n], w[n];
22     qsort(a, n, sizeof(a[0]), cmpx);
23     for (i = 0; i < n; i++) b[i] = a[i], p[i] = 1e16;
24     for (e = 1; e < n; e *= 2) for (l = 0; l + e < n; l += 2*e)
25     {
26         i = k = l, j = m = min(l + e, n), r = min(l + e*2, n);
27         while (i < m && j < r)
28             c[k++] = b[cimag(b[j] - b[i]) < 0.0 ? j++ : i++];
29         while (i < m || j < r) c[k++] = b[j < r ? j++ : i++];
30         for (i = l; i < r; i++) b[i] = c[i];
31         if (p[m] < p[l]) p[l] = p[m], z[l] = z[m], w[l] = w[m];
32         for (k = 0, i = l; i < r; i++) if (fabs(creal(b[i] - a[m])) < p[l])
33             c[k++] = b[i];
34         for (i = 0; i < k; i++)
35             for (j = i + 1; cimag(c[j] - c[i]) < p[l] && j < k; j++)
36                 if (cabs(c[i] - c[j]) < p[l])
37                     p[l] = cabs(c[i] - c[j]), z[l] = c[i], w[l] = c[j];
38     }
39     *x = z[0], *y = w[0];
40     return p[0];
41 }
```

- Hvert endurkvæmt kall er nú $\mathcal{O}(\, n \,)$.
- Svo tímaflækjan á þessari útfærslu er $\mathcal{O}(n \log n)$.
- Þetta má síðan bæta með slembnum reikniritum, en verður annars ekki betra.