Geometrie computațională II

Ștefan Ciobâcă, Dorel Lucanu

Faculty of Computer Science
Alexandru Ioan Cuza University, Iași, Romania
dlucanu@info.uaic.ro

PA 2015/2016

1. Înfășurătoarea convexă
   - Algoritmul naiv
   - Frontiera lui Jarvis (Jarvis's March)
   - Scanarea Graham
   - Înfășurătoarea convexă și Sortarea

Plan

1. Înfășurătoarea convexă
   - Algoritmul naiv
   - Frontiera lui Jarvis (Jarvis's March)
   - Scanarea Graham
   - Înfășurătoarea convexă și Sortarea

Definiția înfășurătorii convexe

Definition

O mulțime de puncte S din plan este convexă dacă pentru orice două puncte $P, Q \in S$, segmentul PQ este complet inclus în S.

Definiția înfășurătorii convexe

Definition

O mulțime de puncte S din plan este convexă dacă pentru orice două puncte $P, Q \in S$, segmentul PQ este complet inclus în S.

Definition

Înfășurătoarea convexă a unei mulțimi de puncte S este cea mai mică muțime convexă care include S.

Definiția înfășurătorii convexe

Definition

O mulțime de puncte S din plan este convexă dacă pentru orice două puncte $P, Q \in S$, segmentul PQ este complet inclus în S.

Definition

Înfășurătoarea convexă a unei mulțimi de puncte S este cea mai mică muțime convexă care include S.

Definition

Înfășurătoarea convexă a unei mulțimi de puncte S este intersecția tuturor mulțimilor convexe care includ S.

Definiția înfășurătorii convexe

Definition

O mulțime de puncte S din plan este convexă dacă pentru orice două puncte $P, Q \in S$, segmentul PQ este complet inclus în S.

Definition

Înfășurătoarea convexă a unei mulțimi de puncte S este cea mai mică muțime convexă care include S.

Definition

Înfășurătoarea convexă a unei mulțimi de puncte S este intersecția tuturor mulțimilor convexe care includ S.

CONVEX HULL
Input O mulțime finită S de puncte din plan.
Output Înfășurătoarea convexă a lui S, $CH(S)$.

Înfășurătoarea convexă: proprietăți

Presupunem S finită.

1. $CH(S)$ este convexă.

Înfășurătoarea convexă: proprietăți

Presupunem S finită.

1. $CH(S)$ este convexă.
2. $CH(S)$ este mărginită.

Înfășurătoarea convexă: proprietăți

Presupunem S finită.

1. $CH(S)$ este convexă.
2. $CH(S)$ este mărginită.
3. Dacă P și Q sunt două puncte din S astfel încât toate celelalte puncte din S se găsesc de aceeași parte a dreptei ce trece prin P și Q, atunci segmentul PQ se află pe frontiera lui $CH(S)$.

Înfășurătoarea convexă: proprietăți

Presupunem S finită.

1. $CH(S)$ este convexă.
2. $CH(S)$ este mărginită.
3. Dacă P și Q sunt două puncte din S astfel încât toate celelalte puncte din S se găsesc de aceeași parte a dreptei ce trece prin P și Q, atunci segmentul PQ se află pe frontiera lui $CH(S)$.
4. Orice punct de frontiera lui $CH(S)$ se găsește pe un segment PQ cu $P, Q \in S$.

Înfășurătoarea convexă: proprietăți

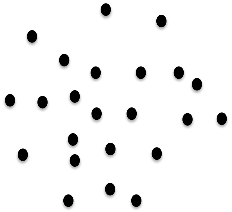
Presupunem S finită.

1. $CH(S)$ este convexă.
2. $CH(S)$ este mărginită.
3. Dacă P și Q sunt două puncte din S astfel încât toate celelalte puncte din S se găsesc de aceeași parte a dreptei ce trece prin P și Q, atunci segmentul PQ se află pe frontiera lui $CH(S)$.
4. Orice punct de frontiera lui $CH(S)$ se găsește pe un segment PQ cu $P, Q \in S$.

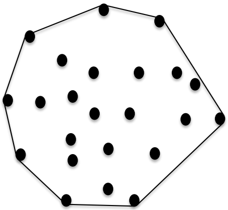
Definition

Dacă S este finită, atunci $CH(S)$ este un poligon convex cu toate vârfurile aparținând la S.

Exemplu de mulțime S



$CH(S)$



Plan

1. Înfășurătoarea convexă
   - Algoritmul naiv
   - Frontiera lui Jarvis (Jarvis's March)
   - Scanarea Graham
   - Înfășurătoarea convexă și Sortarea

Ideea din spatele algoritmului

1. Dacă P și Q sunt două puncte din S astfel încât toate celelalte puncte din S se găsesc de aceeași parte a dreptei ce trece prin P și Q, atunci segmentul PQ se află pe frontiera lui $CH(S)$.

Ideea din spatele algoritmului

1. Dacă P și Q sunt două puncte din S astfel încât toate celelalte puncte din S se găsesc de aceeași parte a dreptei ce trece prin P și Q, atunci segmentul PQ se află pe frontiera lui $CH(S)$.
2. Iterăm prin fiecare pereche P, Q de puncte.

Ideea din spatele algoritmului

1. Dacă P și Q sunt două puncte din S astfel încât toate celelalte puncte din S se găsesc de aceeași parte a dreptei ce trece prin P și Q, atunci segmentul PQ se află pe frontiera lui $CH(S)$.
2. Iterăm prin fiecare pereche P, Q de puncte.

   Dacă toate punctele din $S \setminus \{P, Q\}$ sunt de partea stângă a segmentului orientat PQ, atunci PQ face parte din $CH(S)$.

Ideea din spatele algoritmului

1. Dacă P și Q sunt două puncte din S astfel încât toate celelalte puncte din S se găsesc de aceeași parte a dreptei ce trece prin P și Q, atunci segmentul PQ se află pe frontiera lui $CH(S)$.
2. Iterăm prin fiecare pereche P, Q de puncte.

   Dacă toate punctele din $S \setminus \{P, Q\}$ sunt de partea stângă a segmentului orientat PQ, atunci PQ face parte din $CH(S)$.

   Punctul X este în stânga segmentului PQ dacă `ccw(P, Q, X)` $\leq$ `0`.

Algoritmul naiv

```
@input: o multime S cu n puncte
@output: un poligon convex reprezentand frontiera lui CH(S)
Naiv(S, n) {
  for (i = 0; i < n; ++i) {
    for (j = 0; j < n; ++j) if (i != j) {
      ok = 1;
      for (k = 0; k < n; ++k) {
        if (ccw(S[i], S[j], S[k]) < 0) { ok = 0; }
      }
      if (ok) { next[i] = j; start = i; }
    }
  }
  for (i = start; next[i] != start; i = next[i]) {
    L[l] = S[i]; ++l;
  }
  return L;
```

Analiza algoritmului naiv

1. Corectitudine: bazată pe observația anterioară: dacă P și Q sunt două puncte din S astfel încât toate celelalte puncte din S se găsesc de aceeași parte a dreptei ce trece prin P și Q, atunci segmentul PQ se află pe frontiera lui $CH(S)$.

Analiza algoritmului naiv

1. Corectitudine: bazată pe observația anterioară: dacă P și Q sunt două puncte din S astfel încât toate celelalte puncte din S se găsesc de aceeași parte a dreptei ce trece prin P și Q, atunci segmentul PQ se află pe frontiera lui $CH(S)$.

Analiza algoritmului naiv

1. Corectitudine: bazată pe observația anterioară: dacă P și Q sunt două puncte din S astfel încât toate celelalte puncte din S se găsesc de aceeași parte a dreptei ce trece prin P și Q, atunci segmentul PQ se află pe frontiera lui $CH(S)$.
2. Complexitate timp: $O(n^3)$, complexitate spațiu: $O(n)$.

Analiza algoritmului naiv

1. Corectitudine: bazată pe observația anterioară: dacă P și Q sunt două puncte din S astfel încât toate celelalte puncte din S se găsesc de aceeași parte a dreptei ce trece prin P și Q, atunci segmentul PQ se află pe frontiera lui $CH(S)$.
2. Complexitate timp: $O(n^3)$, complexitate spațiu: $O(n)$.
3. De ce analizăm toate perechile de puncte $(S[i], S[j])$ cu $i \neq j$ și nu doar perechile $(S[i], S[j])$ cu $i < j$?

Plan

1. Înfășurătoarea convexă
   - Algoritmul naiv
   - Frontiera lui Jarvis (Jarvis's March)
   - Scanarea Graham
   - Înfășurătoarea convexă și Sortarea

Descriere

Cunoscută și sub numele de "gift wrapping method".

Descriere

Cunoscută și sub numele de "gift wrapping method".
Idee:

- Se baleiază (mătură) planul cu o dreaptă d până atinge un punct $P_0 \in S$. P_0 se află pe frontiera lui $CH(S)$.

Descriere

Cunoscută și sub numele de "gift wrapping method".
Idee:

- Se baleiază (mătură) planul cu o dreaptă d până atinge un punct $P_0 \in S$. P_0 se află pe frontiera lui $CH(S)$.
- Se rotește d, să zicem că în sensul CCW, până atinge al doilea punct P_1. P_1 se află pe frontiera lui $CH(S)$.

Descriere

Cunoscută și sub numele de "gift wrapping method".
Idee:

- Se baleiază (mătură) planul cu o dreaptă d până atinge un punct $P_0 \in S$. P_0 se află pe frontiera lui $CH(S)$.
- Se rotește d, să zicem că în sensul CCW, până atinge al doilea punct P_1. P_1 se află pe frontiera lui $CH(S)$.
- Se rotește dreapta P_0P_1, același sens, până atinge al treilea punct P_2. P_2 se află pe frontiera lui $CH(S)$.

Descriere

Cunoscută și sub numele de "gift wrapping method".
Idee:

- Se baleiază (mătură) planul cu o dreaptă d până atinge un punct $P_0 \in S$. P_0 se află pe frontiera lui $CH(S)$.
- Se rotește d, să zicem că în sensul CCW, până atinge al doilea punct P_1. P_1 se află pe frontiera lui $CH(S)$.
- Se rotește dreapta $P_0 P_1$, același sens, până atinge al treilea punct P_2. P_2 se află pe frontiera lui $CH(S)$.
- Se rotește dreapta $P_1 P_2$, același sens, până atinge al ...

Descriere

Cunoscută și sub numele de "gift wrapping method".
Idee:

- Se baleiază (mătură) planul cu o dreaptă d până atinge un punct $P_0 \in S$. P_0 se află pe frontiera lui $CH(S)$.
- Se rotește d, să zicem că în sensul CCW, până atinge al doilea punct P_1. P_1 se află pe frontiera lui $CH(S)$.
- Se rotește dreapta P_0P_1, același sens, până atinge al treilea punct P_2. P_2 se află pe frontiera lui $CH(S)$.
- Se rotește dreapta P_1P_2, același sens, până atinge al ...
- În final punctele atinse formează frontiera lui $CH(S)$, care e un poligon convex.

Descriere

Cunoscută și sub numele de "gift wrapping method".
Idee:

- Se baleiază (mătură) planul cu o dreaptă d până atinge un punct $P_0 \in S$. P_0 se află pe frontiera lui $CH(S)$.
- Se rotește d, să zicem că în sensul CCW, până atinge al doilea punct P_1. P_1 se află pe frontiera lui $CH(S)$.
- Se rotește dreapta P_0P_1, același sens, până atinge al treilea punct P_2. P_2 se află pe frontiera lui $CH(S)$.
- Se rotește dreapta P_1P_2, același sens, până atinge al ...
- În final punctele atinse formează frontiera lui $CH(S)$, care e un poligon convex.

Observație cheie: la pasul i, unghiul $\widehat{P_{i-1}P_iP_{i+1}}$ este cel mai mare pe care îl poate forma segmentul $P_{i-1}P_i$ cu un alt punct din S.

Primitive

Presupunem $|S| = n$.
`smallestY(S)` - întoarce punctul din S cu cel mai mic y; dacă există mai multe, atunci cel mai din stânga.
Timp: $O(n)$.

Primitive

Presupunem $|S| = n$.
`smallestY(S)` - întoarce punctul din S cu cel mai mic y; dacă există mai multe, atunci cel mai din stânga.
Timp: $O(n)$.

`smallestSlope(S, P)` - întoarce punctul Q din S a.î. PQ are cea mai mică pantă în raport cu axa x.
Timp: $O(n)$

Primitive

Presupunem $|S| = n$.
`smallestY(S)` - întoarce punctul din S cu cel mai mic y; dacă există mai multe, atunci cel mai din stânga.
Timp: $O(n)$.

`smallestSlope(S, P)` - întoarce punctul Q din S a.î. PQ are cea mai mică pantă în raport cu axa x.
Timp: $O(n)$

`largestAngle(S, P, Q)` - întoarce punctul R din S a.î. $\widehat{PQR}$ este cel mai mare.
Timp: $O(n)$

Primitive

Presupunem $|S| = n$.
`smallestY(S)` - întoarce punctul din S cu cel mai mic y; dacă există mai multe, atunci cel mai din stânga.
Timp: $O(n)$.

`smallestSlope(S, P)` - întoarce punctul Q din S a.î. PQ are cea mai mică pantă în raport cu axa x.
Timp: $O(n)$

`largestAngle(S, P, Q)` - întoarce punctul R din S a.î. $\widehat{PQR}$ este cel mai mare.
Timp: $O(n)$

Exercițiu. Să se descrie în Alk cele trei operații de mai sus.

Algoritmul

```
@input: o multime S cu n puncte
@output: un poligon convex reprezentand frontiera lui CH(S)
CHJarvis(S, n) {
  L[0] = smallestY(S);
  L[1] = smallestSlope(S, L[0]);
  i = 1;
  while (L[i] != L[0]) {
    L[i+1] = largestAngle(S, L[i-1], L[i]);
    i = i + 1;
  }
  return L;
}
```

Analiza algoritmului

- Invariantul instrucțiunii `while`: `L[0], ..., L[i]` se află pe frontiera lui $CH(S)$. Validitatea acestui invariant este menținută de apelul `largestAngle(S, L[i-1], L[i])` care asigură că toate punctele lui S se află la intersecția semiplanelor $(L[i-1]L[i], L[i+1])$ și $(L[i]L[i+1], L[i-1])$.

Analiza algoritmului

- Invariantul instrucțiunii `while`: `L[0], ..., L[i]` se află pe frontiera lui $CH(S)$. Validitatea acestui invariant este menținută de apelul `largestAngle(S, L[i-1], L[i])` care asigură că toate punctele lui S se află la intersecția semiplanelor $(L[i-1]L[i], L[i+1])$ și $(L[i]L[i+1], L[i-1])$.
- Complexitate timp: $O(n) + O(n) + O(h) \cdot O(n) = O(h \cdot n) = O(n^2)$, complexitate spațiu $O(n)$.

Analiza algoritmului

- Invariantul instrucțiunii `while`: `L[0], ..., L[i]` se află pe frontiera lui $CH(S)$. Validitatea acestui invariant este menținută de apelul `largestAngle(S, L[i-1], L[i])` care asigură că toate punctele lui S se află la intersecția semiplanelor $(L[i-1]L[i], L[i+1])$ și $(L[i]L[i+1], L[i-1])$.
- Complexitate timp: $O(n) + O(n) + O(h) \cdot O(n) = O(h \cdot n) = O(n^2)$, complexitate spațiu $O(n)$.
- Timpul de rulare $O(h \cdot n)$ depinde de dimensiunea h a `output`-ului. Acești algoritmi se numesc *output-sensitive*.

Plan

1. Înfășurătoarea convexă
   - Algoritmul naiv
   - Frontiera lui Jarvis (Jarvis's March)
   - Scanarea Graham
   - Înfășurătoarea convexă și Sortarea

Descriere

1. Se începe cu un punct P de coordonată Y minimă (alternativ se poate folosi orice punct din interiorul $CH(S)$).

Descriere

1. Se începe cu un punct P de coordonată Y minimă (alternativ se poate folosi orice punct din interiorul $CH(S)$).
2. Se sortează lexicografic S după coordonatele polare (θ, ρ), presupunând că originea este P.

Descriere

1. Se începe cu un punct P de coordonată Y minimă (alternativ se poate folosi orice punct din interiorul $CH(S)$).
2. Se sortează lexicografic S după coordonatele polare (θ, ρ), presupunând că originea este P.
3. inițial L are lungimea $l = 2$: $L[0] = S[0] = P$, $L[1] = S[1]$

Descriere

1. Se începe cu un punct P de coordonată Y minimă (alternativ se poate folosi orice punct din interiorul $CH(S)$).
2. Se sortează lexicografic S după coordonatele polare (θ, ρ), presupunând că originea este P.
3. inițial L are lungimea $l = 2$: $L[0] = S[0] = P$, $L[1] = S[1]$
4. Începând cu $p = 2$ adaugă punctul $S[p]$ la înfășurătoare:

Descriere

1. Se începe cu un punct P de coordonată Y minimă (alternativ se poate folosi orice punct din interiorul $CH(S)$).
2. Se sortează lexicografic S după coordonatele polare (θ, ρ), presupunând că originea este P.
3. inițial L are lungimea $l = 2$: $L[0] = S[0] = P$, $L[1] = S[1]$
4. Începând cu $p = 2$ adaugă punctul $S[p]$ la înfășurătoare:
   - cât timp $L[l-2], L[L-1], S[p]$ sunt în ordine antitrigonometrică șterge ultimul element din lista L

Descriere

1. Se începe cu un punct P de coordonată Y minimă (alternativ se poate folosi orice punct din interiorul $CH(S)$).
2. Se sortează lexicografic S după coordonatele polare (θ, ρ), presupunând că originea este P.
3. inițial L are lungimea $l = 2$: $L[0] = S[0] = P$, $L[1] = S[1]$
4. Începând cu $p = 2$ adaugă punctul $S[p]$ la înfășurătoare:
   - cât timp $L[l - 2], L[L - 1], S[p]$ sunt în ordine antitrigonometrică șterge ultimul element din lista L
   - adaugă $S[p]$ la L și trece la următorul p

Primitive

`smallestY` întoarce punctul cu cel mai mic Y (timp: $O(n)$).

Primitive

`smallestY` întoarce punctul cu cel mai mic Y (timp: $O(n)$).

`polarSorted(S, n, P)` - întoarce lista liniară conținând elementele lui S sortate lexicografic după coordonatele polare (θ, ρ), calculate cu originea în P.
Timp: $O(n \log n)$.

Primitive

`smallestY` întoarce punctul cu cel mai mic Y (timp: $O(n)$).

`polarSorted(S, n, P)` - întoarce lista liniară conținând elementele lui S sortate lexicografic după coordonatele polare (θ, ρ), calculate cu originea în P.
Timp: $O(n \log n)$.

`ccw(P, Q, R)` - întoarce -1, 0 sau 1, după cum P, Q, R sunt în sens antitrigonometric, colineare sau trigonometric.

Scanarea Graham

```
@input: o multime S cu n puncte
@output: un poligon convex reprezentand frontiera lui CH(S)
CHGraham(S, n) {
  P = smallestY(S);
  S = polarSorted(S, n, P); // presupun S[0] = P
  L[0] = S[0]; L[1] = S[1]; l = 2;
  for (i = 2; i < n; ++i) {
    while (ccw(L[l - 2], L[l - 1], S[i]) < 0) {
      --l; // "sterge" ultimul element din L
    }
    L[l] = S[i];
    ++l;
  }
  return L;
 }
```

Analiza

1. Corectitudinea algoritmului este asigurată de următorul invariant:
   - După execuția iterației i a buclei `for`, lista L conține înfășurătoarea convexă a primelor i puncte din S.
2. Complexitate timp: $O(n \log n)$ (dominat de sortare – scanarea se face în timpul $O(n)$), complexitate spațiu: $O(n)$.

Sortarea după coordonatele polare

```
@input: două puncte A, B și "originea" P
        aî A, B sunt "deasupra" lui P
@output: dacă A apare înaintea lui B în ordinea
         unghiurilor polare față de P
smaller(P, A, B) {
  A.theta = atan2(A.x - P.x, A.y - P.y);
  A.rho   = sqrt(sqr(A.x - P.x), sqr(A.y - P.y));
  B.theta = atan2(B.x - P.x, B.y - P.y);
  B.rho   = sqrt(sqr(B.x - P.x), sqr(B.y - P.y));
  return (A.theta < B.theta) ||
         (A.theta == B.theta && A.rho < B.rho);
}
```

Sortarea după coordonatele polare

```
@input: două puncte A, B și "originea" P
        aî A, B sunt "deasupra" lui P
@output: dacă A apare înaintea lui B în ordinea
         unghiurilor polare față de P
smaller(P, A, B) {
  A.theta = atan2(A.x - P.x, A.y - P.y);
  A.rho   = sqrt(sqr(A.x - P.x), sqr(A.y - P.y));
  B.theta = atan2(B.x - P.x, B.y - P.y);
  B.rho   = sqrt(sqr(B.x - P.x), sqr(B.y - P.y));
  return (A.theta < B.theta) ||
         (A.theta == B.theta && A.rho < B.rho);
}
```

Dezavantaj major: posibile erori de calcul datorate rotunjirilor.

Sortarea după coordonatele polare

```
@input: două puncte A, B și "originea" P
        aî A, B sunt "deasupra" lui P
@output: dacă A apare înaintea lui B în ordinea
         unghiurilor polare față de P
smaller(P, A, B) {
  return ccw(P, A, B) == 1;
}
```

Sortarea după coordonatele polare

```
@input: două puncte A, B și "originea" P
        aî A, B sunt "deasupra" lui P
@output: dacă A apare înaintea lui B în ordinea
         unghiurilor polare față de P
smaller(P, A, B) {
  return ccw(P, A, B) == 1;
}
```

Avantaj: nu mai apar erori de calcul.

Plan

1. Înfășurătoarea convexă
   - Algoritmul naiv
   - Frontiera lui Jarvis (Jarvis's March)
   - Scanarea Graham
   - Înfășurătoarea convexă și Sortarea

Modelul de calcul al arborilor de decizie algebrici

Sunt generalizări ale arborilor de decizie pentru sortare și arborilor de decizie pentru căutare.

Modelul de calcul al arborilor de decizie algebrici

Sunt generalizări ale arborilor de decizie pentru sortare și arborilor de decizie pentru căutare.

Ideea de bază: în loc de comparații $s[i]?s[j]$ sau $x?s[m]$ se consideră comparații $f(x_1, \ldots, x_n)?0$, unde f este o funcție polinomială cu n argumente.

Modelul de calcul al arborilor de decizie algebrici

Sunt generalizări ale arborilor de decizie pentru sortare și arborilor de decizie pentru căutare.

Ideea de bază: în loc de comparații $s[i]?s[j]$ sau $x?s[m]$ se consideră comparații $f(x_1, \ldots, x_n)?0$, unde f este o funcție polinomială cu n argumente.

Exemple de polinoame F: $\texttt{ccw}(A, B, C)$, $\texttt{dist}(A, B)$, $\texttt{dist}(d, P)$

Modelul de calcul al arborilor de decizie algebrici

Sunt generalizări ale arborilor de decizie pentru sortare și arborilor de decizie pentru căutare.

Ideea de bază: în loc de comparații $s[i]?s[j]$ sau $x?s[m]$ se consideră comparații $f(x_1, \ldots, x_n)?0$, unde f este o funcție polinomială cu n argumente.

Exemple de polinoame F: $\texttt{ccw}(A, B, C)$, $\texttt{dist}(A, B)$, $\texttt{dist}(d, P)$
NB. $\texttt{dist}(A, B)$ și $\texttt{dist}(d, P)$ nu sunt chiar polinoame, dar se poate lua pătratul funcțiilor distanță, care nu schimbă comparația cu zero și sunt polinoame.

Modelul de calcul al arborilor de decizie algebrici

Sunt generalizări ale arborilor de decizie pentru sortare și arborilor de decizie pentru căutare.

Ideea de bază: în loc de comparații $s[i]?s[j]$ sau $x?s[m]$ se consideră comparații $f(x_1, \ldots, x_n)?0$, unde f este o funcție polinomială cu n argumente.

Exemple de polinoame F: $\texttt{ccw}(A, B, C)$, $\texttt{dist}(A, B)$, $\texttt{dist}(d, P)$
NB. $\texttt{dist}(A, B)$ și $\texttt{dist}(d, P)$ nu sunt chiar polinoame, dar se poate lua pătratul funcțiilor distanță, care nu schimbă comparația cu zero și sunt polinoame.

Ca și în cazul sortării, se pot considera doar arbori binari.

Modelul de calcul al arborilor de decizie algebrici

Sunt generalizări ale arborilor de decizie pentru sortare și arborilor de decizie pentru căutare.

Ideea de bază: în loc de comparații $s[i]?s[j]$ sau $x?s[m]$ se consideră comparații $f(x_1, \ldots, x_n)?0$, unde f este o funcție polinomială cu n argumente.

Exemple de polinoame F: $\texttt{ccw}(A, B, C)$, $\texttt{dist}(A, B)$, $\texttt{dist}(d, P)$
NB. $\texttt{dist}(A, B)$ și $\texttt{dist}(d, P)$ nu sunt chiar polinoame, dar se poate lua pătratul funcțiilor distanță, care nu schimbă comparația cu zero și sunt polinoame.

Ca și în cazul sortării, se pot considera doar arbori binari.

Algoritmii de geometrie computațională pot fi reprezentați ca arbori de decizie algebrici.

Modelul de calcul al arborilor de decizie algebrici

Sunt generalizări ale arborilor de decizie pentru sortare și arborilor de decizie pentru căutare.

Ideea de bază: în loc de comparații $s[i]?s[j]$ sau $x?s[m]$ se consideră comparații $f(x_1, \ldots, x_n)?0$, unde f este o funcție polinomială cu n argumente.

Exemple de polinoame F: $\texttt{ccw}(A, B, C)$, $\texttt{dist}(A, B)$, $\texttt{dist}(d, P)$
NB. $\texttt{dist}(A, B)$ și $\texttt{dist}(d, P)$ nu sunt chiar polinoame, dar se poate lua pătratul funcțiilor distanță, care nu schimbă comparația cu zero și sunt polinoame.

Ca și în cazul sortării, se pot considera doar arbori binari.

Algoritmii de geometrie computațională pot fi reprezentați ca arbori de decizie algebrici.
Reducerile de pe slide-urile următoare se referă la acest model de calcul.

SORT $\propto$ CONVEX HULL

Theorem

SORT $\propto_n$ CONVEX HULL.

SORT $\propto$ CONVEX HULL

Theorem

SORT $\propto_n$ CONVEX HULL.

$S = \{x_0, \ldots, x_n\}$
Presupunem $x_i > 0$, $i = 1, \ldots, n$ (în caz contrar se face o "translatare").

SORT $\propto$ CONVEX HULL

Theorem

SORT $\propto_n$ CONVEX HULL.

$S = \{x_0, \ldots, x_n\}$
Presupunem $x_i > 0$, $i = 1, \ldots, n$ (în caz contrar se face o "translatare").

Se face transformarea: $S \mapsto S' = \{(x, x^2) \mid x \in S\}$

SORT $\propto$ CONVEX HULL

Theorem

SORT $\propto_n$ CONVEX HULL.

$S = \{x_0, \ldots, x_n\}$
Presupunem $x_i > 0$, $i = 1, \ldots, n$ (în caz contrar se face o "translatare").

Se face transformarea: $S \mapsto S' = \{(x, x^2) \mid x \in S\}$

Graficul lui $y = x^2$ este o parabolă. $CH(S')$ conține toate punctele din S', ordonate după coordonata x.

SORT $\propto$ CONVEX HULL

Theorem

SORT $\propto_n$ CONVEX HULL.

$S = \{x_0, \ldots, x_n\}$
Presupunem $x_i > 0$, $i = 1, \ldots, n$ (în caz contrar se face o "translatare").

Se face transformarea: $S \mapsto S' = \{(x, x^2) \mid x \in S\}$

Graficul lui $y = x^2$ este o parabolă. $CH(S')$ conține toate punctele din S', ordonate după coordonata x.

Corollary

CONVEX HULL are complexitatea timp în cazul cel mai nefavorabil $\Omega(n \log n)$.

CONVEX HULL $\propto$ SORT

Theorem

CONVEX HULL $\propto_n$ SORT .

CONVEX HULL $\propto$ SORT

Theorem

CONVEX HULL $\propto_n$ SORT .

Scanarea Graham.

CONVEX HULL $\propto$ SORT

Theorem

CONVEX HULL $\propto_n$ SORT .

Scanarea Graham.

Corolar

Scanarea Graham este un algoritm optim pentru problema înfășurătorii convexe.