Distanțe

Capitolul 19

- **Preliminarii**
- **Distanța minimă între două puncte**
- **Distanța maximă între două puncte**
- **Rezumat**
- **Implementări sugerate**
- **Probleme propuse**
- **Soluțiile problemelor**

Acest capitol tratează subiectul distanțelor dintre puncte în plan. Vom prezenta mai întâi cele mai utilizate tipuri de distanțe, urmând ca ulterior să descriem algoritmi eficienți care pot fi utilizați pentru a determina, pentru o colecție de puncte, distanța minimă şi distanța maximă între două puncte.

19.1. Preliminarii

În planul euclidian, distanța dintre două puncte de coordonate (x_1, y_1) şi (x_2, y_2) se defineşte ca fiind lungimea segmentului care le uneşte. Aşadar, pentru calculul unei astfel de distanțe este suficient să folosim cunoscuta formulă:

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} .$$

Această formulă poate fi extinsă şi pentru spațiul tridimensional; aşadar, pentru două puncte de coordonate (x_1, y_1, z_1) şi (x_2, y_2, z_2), distanța dintre ele poate fi calculată pe baza formulei:

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2} .$$

Aceasta este numită distanța euclidiană dintre două puncte date. Există multe alte tipuri de distanțe, definite în diverse moduri. Cea mai des utilizată dintre acestea este *distanța Manhattan*, a cărei denumire este inspirată de la configurația străzilor din cartierul new-yorkez cu acelaşi nume.

În Manhattan, majoritatea străzilor formează doar unghiuri drepte. Aşadar, dacă alegem un sistem de coordonate, străzile vor fi paralele cu cele două axe. Din acest motiv, pentru a ajunge dintr-un punct în altul ne putem deplasa doar paralel cu axele de coordonate.

Distanţa Manhatten reprezintă lungimea totală a drumului care trebuie parcurs de la un punct la altul dacă ne deplasăm doar paralel cu axele de coordonate. Practic, drumul va fi format dintr-un segment orizontal şi unul vertical.

În cazul în care cele două puncte au coordonata orizontală egală, atunci segmentul vertical degenerează într-un punct (are lungimea 0). Similar, dacă cele două puncte au coordonata verticală egală, atunci segmentul orizontal degenerează într-un punct.

Lungimea segmentului vertical este egală cu diferenţa dintre coordonatele verticale ale celor două puncte, iar cea a segmentului orizontal este egală cu diferenţa coordonatelor orizontale. Formula de calcul pentru distanţa Manhattan este, aşadar, următoarea:

$$d = |x_1 - x_2| + |y_1 - y_2|.$$

Bineînţeles, şi această distanţă poate fi generalizată pentru spaţiul tridimensional, formula devenind:

$$d = |x_1 - x_2| + |y_1 - y_2| + |z_1 - z_2|.$$

Aşadar, în spaţiul tridimensional vom avea trei segmente, fiecare paralel cu unul dintre planuri.

Mai mult, distanţele pot fi generalizate pentru spaţii cu un număr de dimensiuni oricât de mare. Deşi, utilitatea practică a unor astfel de formule este redusă, vom prezenta formulele generale de calcul atât pentru distanţa Manhattan, cât şi pentru cea euclidiană.

Într-un spaţiu n-dimensional, un punct va fi identificat prin n coordonate (proiecţiile pe cele n axe ale spaţiului n-dimensional). Aşadar, pentru două puncte vom avea $P_1 = (x_{11}, x_{12}, \ldots, x_{1n})$ şi $P_2 = (x_{21}, x_{22}, \ldots, x_{2n})$.

Distanţa Manhattan dintre cele două puncte va fi:

$$d = |x_{11} - x_{21}| + |x_{12} - x_{22}| + \cdots + |x_{1n} - x_{2n}|,$$

iar distanţa euclidiană va fi:

$$d = \sqrt{(x_{11} - x_{21})^2 + (x_{12} - x_{22})^2 + \cdots + (x_{1n} - x_{2n})^2}.$$

19.2. Distanţa minimă dintre două puncte

Considerăm că se cunosc coordonatele a N puncte în plan. Se doreşte identificarea unei perechi de puncte între care se află cea mai mică distanţă. Cu alte cuvinte, dorim identificarea unei perechi de puncte, astfel încât distanţa dintre aceste două puncte să fie mai mică sau egală cu distanţa dintre oricare pereche de puncte.

În cadrul acestei secţiuni vom prezenta algoritmul de rezolvare simplu, dar ineficient, precum şi un algoritm eficient, bazat pe metoda divide et impera.

19.2.1. Algoritmul ineficient

Cea mai simplă metodă de rezolvare a acestei probleme constă în determinarea distanțelor dintre toate perechile de puncte (vor fi $N \cdot (N - 1) / 2$ astfel de perechi) şi alegerea celei mai mici dintre ele.

Aşadar, problema se reduce la o simplă determinare a unei valori minime dintre $N \cdot (N - 1) / 2$ valori care sunt calculate.

Evident, ordinul de complexitate al acestui algoritm este $O(N^2)$ deoarece trebuie să calculăm $N \cdot (N - 1) / 2$ distanțe şi să aplicăm un algoritm de determinare a minimului care necesită efectuarea unei comparații pentru fiecare distanță, cu excepția primeia.

19.2.2. Algoritmul eficient

Pentru a obține un algoritm cu o complexitate mai bună vom folosi metoda *divide et impera*. Un apel recursiv al algoritmului va avea ca intrare o submulțime P a mulțimii punctelor şi două şiruri X şi Y. Punctele din şirul X sunt ordonate în aşa fel încât coordonatele orizontale să fie în ordine crescătoare. De asemenea, punctele din şirul Y sunt ordonate în aşa fel încât coordonatele verticale să fie crescătoare.

Pentru a obține ordinul de complexitate dorit, nu ne permitem să efectuăm sortări la fiecare apel recursiv. Pentru a evita acest lucru vom folosi o metodă numită presortare, cu ajutorul căreia vom menține şirurile ordonate fără a le sorta efectiv la fiecare apel recursiv.

Un apel recursiv cu intrările P, X şi Y va verifica mai întâi dacă în mulțimea P există mai mult de trei puncte. Dacă avem cel mult trei puncte, atunci vom verifica distanțele dintre toate perechile de puncte (avem o pereche dacă sunt două puncte şi trei perechi dacă sunt trei puncte; nu este posibil să avem un punct) şi o vom alege pe cea minimă. Pentru mulțimi cu mai mult de trei puncte vom aplica cele trei etape ale algoritmilor *divide et impera*: *împarte*, *stăpâneşte* şi *combină*.

Pentru prima etapă vom determina o dreaptă verticală d care împarte mulțimea punctelor P în două submulțimi P_S şi P_D, fiecare conținând jumătate din punctele mulțimii P. Dacă numărul de puncte din P este impar, atunci una dintre mulțimi va conține cu un punct mai mult decât cealaltă. Toate punctele din P_S vor fi pe dreapta d sau în stânga acesteia şi toate punctele din P_D sunt pe dreapta d sau în dreapta acesteia. Şirul X este împărțit în şirurile X_S şi X_D care conțin punctele din P_S, respectiv P_D, ordonate crescător după coordonata orizontală. Analog, şirul Y este împărțit în şirurile Y_S şi Y_D care conțin punctele din P_S, respectiv P_D, ordonate crescător după coordonata verticală.

La cea de-a doua etapă, având împărțită mulțimea P în submulțimile P_S şi P_D, vom efectua două apeluri recursive, unul pentru a găsi cea mai apropiată pereche de puncte din P_S şi celălalt pentru a găsi cea mai apropiată pereche din P_D. Pentru primul apel vom folosi ca intrări mulțimea P_S şi şirurile X_S şi Y_S, iar pentru al doilea mulțimea P_D şi şirurile X_D şi Y_D. Fie d_S şi d_D valorile returnate în urma celor două apeluri şi D minimul acestora.

La cea de-a treia etapă vom şti că cea mai apropiată pereche de puncte din mulțimea P este, fie cea aflată la distanța D, fie o altă pereche de puncte, unul aflat în mulțimea P_S şi celălalt aflat în mulțimea P_D. Va trebui să verificăm dacă există o astfel de pereche aflată la o distanță mai mică decât D. Observăm că, dacă există o astfel de pereche, atunci aceste puncte se află la o distanță de cel mult D față de dreapta d. Ca urmare, ele trebuie să se afle într-o regiune de lățime $2 \cdot D$ centrată în jurul dreptei d. Pentru a identifica o astfel de pereche vom proceda astfel:

- Construim un şir Y' care conține toate punctele din şirul Y, care se află în interiorul regiunii de lățime $2 \cdot D$. Acest şir va fi sortat crescător după coordonata verticală a punctelor.
- Pentru fiecare punct p din şirul Y' vom încerca să găsim punctele din Y' care se află la o distanță cel mult D față de p. Aşa cum vom arăta ulterior, este necesar să fie considerate doar şapte puncte din Y', care urmează după p. Vom calcula distanțele de la p la cele şapte puncte şi vom reține distanța minimă D' a perechii celei mai apropiate.
- Dacă $D' < D$, atunci regiunea verticală conține o pereche mai apropiată decât cea găsită prin apelurile recursive. În acest caz este returnată această pereche şi distanța D' dintre cele două puncte care o formează. Altfel, este returnată perechea cea mai apropiată găsită prin apelurile recursive, împreună cu distanța sa D.

În această descriere a algoritmului am omis câteva detalii de implementare care sunt necesare pentru a obține ordinul de complexitate $O(n \cdot \log n)$. Vom reveni asupra acestora după ce vom demonstra corectitudinea algoritmului.

19.2.3. Corectitudinea algoritmului

Există doar două aspecte care pot fi luate în considerare cu privire la corectitudinea algoritmului. Primul constă în oprirea apelurilor recursive atunci când avem mai puțin de patru puncte, deoarece trebuie să ne asigurăm că nu vom încerca niciodată să împărțim o mulțime care conține un singur punct.

Al doilea aspect constă în demonstrarea faptului că avem nevoie de verificarea a doar şapte puncte pentru fiecare punct p din Y'. Această proprietate este demonstrată în continuare.

Să presupunem că la un anumit nivel al recursivității, punctele cele mai apropiate se află unul în mulțimea P_S şi unul în mulțimea P_D. Fie acestea p_S şi p_D; distanța D' dintre acestea este strict mai mică decât D. Punctul p_S trebuie să fie pe dreapta d sau în stânga ei, iar punctul p_D trebuie să fie pe dreapta d sau în dreapta ei; cele două puncte trebuie să se afle la o distanță de cel mult D unități față de dreapta d. Mai mult, distanța pe verticală dintre p_S şi p_D trebuie să fie cel mult D. În concluzie, p_S şi p_D se află într-un dreptunghi de lungime $2 \cdot D$ şi înălțime D, centrat pe dreapta d.

Vom arăta în continuare că există cel mult opt puncte care se pot afla în interiorul unui astfel de dreptunghi. Dreapta d împarte dreptunghiul în două pătrate cu latura de

lungime D. Să considerăm acum pătratul din stânga dreptei d. Deoarece toate punctele din P_S se află la o distanță de cel puțin D față de p_S (altfel după apelul recursiv ar fi fost returnată o altă distanță mai mică), cel mult patru puncte pot fi situate în interiorul acestui pătrat. În mod analog putem demonstra că cel mult patru puncte se pot situa în pătratul din dreapta dreptei d. În concluzie, cel mult opt puncte din P se pot situa în interiorul dreptunghiului. Din acest motiv, este suficient să verificăm cel mult şapte puncte care urmează după fiecare punct din Y'.

Putem presupune, fără a reduce generalitatea, că punctul p_S se află înaintea punctului p_D în şirul Y'. În acest caz, chiar dacă p_S apare cât mai devreme în şirul Y' şi p_D apare cât mai târziu în acest şir, p_D se află pe una dintre cele şapte poziții care urmează după p_S.

Cu aceasta am demonstrat corectitudinea algoritmului de determinare a celei mai apropiate perechi de puncte.

19.2.4. Detalii de implementare

Pentru a obține complexitatea dorită trebuie să ne asigurăm că punctele din şirurile X_S, X_D, Y_S şi Y_D care sunt folosite în apelurile recursive, sunt sortate după coordonatele orizontale, respectiv verticale. De asemenea, punctele din şirul Y' trebuie să fie sortate după coordonatele verticale. Observăm că dacă şirul punctelor din mulțimea P este deja sortat, împărțirea mulțimii P în submulțimile P_S şi P_D se realizează foarte uşor în timp liniar.

Observația cheie a rezolvării acestei probleme este aceea că dorim să obținem un subşir sortat al unui şir sortat. Fie un apel particular, având ca date de intrare mulțimea P şi şirul Y sortate după coordonata verticală. După ce am împărțit mulțimea P în submulțimile P_S şi P_D, trebuie să formăm în timp liniar şirurile Y_S şi Y_D, care vor fi sortate după coordonata verticală.

Pentru aceasta vom examina, în ordine, punctele din şirul Y. Dacă un anumit punct se află în mulțimea P_S, atunci îl adăugăm la sfârşitul şirului Y_S, iar dacă se află în mulțimea P_D, atunci îl adăugăm la sfârşitul şirului Y_D. Şirurile X_S, X_D şi Y' sunt create în mod analog.

Mai rămâne problema sortării inițiale a punctelor. Vom face o simplă presortare a lor, adică le vom sorta, o dată, înainte de primul apel recursiv. Şirurile sortate sunt transmise ca parametri în primul apel şi apoi sunt reduse pe măsură ce se avansează în recursivitate. Presortarea poate fi realizată într-un timp $O(n \cdot \log n)$.

19.2.5. Analiza complexității

Vom analiza acum complexitatea algoritmului divide et impera utilizat. Notăm cu $T(n)$ ordinul de complexitate al acestuia. La fiecare pas vom avea două apeluri recursive ale aceluiaşi algoritm pentru jumătate din puncte la care se adaugă câteva operații care pot fi efectuate în timp liniar. Putem scrie relația de recurență pentru ordinul de complexi-

tate al algoritmului ca fiind $T(n) = 2 \cdot T(n / 2) + O(n)$. Această formulă este valabilă pentru $n > 3$, deoarece ne oprim din recursivitate dacă nu avem cel puțin patru puncte. Pentru $n < 3$ ordinul este $O(1)$, deoarece sunt efectuate un număr constant de operații.

Din formula recursivă a ordinului de complexitate rezultă că acesta este $O(n \cdot \log n)$. Mai trebuie adăugat şi algoritmul de presortare, dar ordinul de complexitate nu se schimbă deoarece există metode de sortare care funcționează într-un timp de ordinul $O(n \cdot \log n)$.

19.3. Distanța minimă dintre două puncte

La fel ca şi în cazul problemei determinării distanței minime, considerăm că se cunosc coordonatele a N puncte în plan. De data aceasta, se doreşte identificarea unei perechi de puncte între care se află cea mai mare distanță. Ca urmare, dorim identificarea unei perechi de puncte, astfel încât distanța dintre aceste două puncte să fie mai mare sau egală cu distanța dintre oricare pereche de puncte.

Din nou, vom prezenta algoritmul de rezolvare simplu, dar ineficient, precum şi un algoritm eficient, bazat pe determinarea unei înfăşurători convexe.

19.3.1. Algoritmul ineficient

O rezolvarea ineficientă este, practic, identică celei pentru problema distanței minime. Singura modificare constă în faptul că vom determina un maxim şi nu un minim. Din nou, cea mai simplă metodă de rezolvare a acestei probleme constă în determinarea distanțelor dintre toate perechile de puncte (vor fi tot $N \cdot (N - 1) / 2$ astfel de perechi) şi alegerea, de data aceasta, a celei mai mari dintre ele.

Aşadar, problema se reduce la o simplă determinare a unei valori maxime dintre $N \cdot (N - 1) / 2$ valori care sunt calculate.

Evident, ordinul de complexitate al acestui algoritm este tot $O(N^2)$, motivele fiind aceleaşi ca în cazul problemei de determinare a distanței minime.

19.3.2. Algoritmul eficient

Pentru început, este uşor de observat faptul că cele două puncte se află, obligatoriu, pe înfăşurătoarea convexă a mulțimii de puncte.

Vom demonstra această afirmație prin metoda reducerii la absurd. Să presupunem că unul dintre cele două puncte nu se află pe înfăşurătoarea convexă, aşadar se află în interiorul acesteia. Vom nota celălalt punct prin P, iar acest punct prin Q. Dacă vom prelungi segmentul care uneşte punctele P şi Q, atunci acesta va intersecta o latură a înfăşurătorii convexe sau va trece printr-un alt punct al acesteia.

În acest din urmă caz se observă imediat că distanța de la P la Q este mai mică decât distanța de la P la punctul respectiv, deci punctele P şi Q nu se pot afla la distanța maximă.

Pentru cel de-al doilea caz vom considera că latura intersectată are ca extremități punctele R şi S şi prelungirea segmentului care uneşte punctele P şi Q intersectează latura respectivă în punctul T.

Pentru început se observă faptul că lungimea segmentului care uneşte punctele P şi T este mai mare decât cea care uneşte punctele P şi Q.

Vom considera acum triunghiul PRS (determinat de vârfurile P, R şi S). Punctul T se va afla pe latura RS a acestui triunghi. Există o teoremă care afirmă că cel puțin una dintre laturile PR şi PS are lungimea mai mare decât segmentul PT. Ca urmare una dintre laturile triunghiului va fi mai lungă decât segmentul PT care, la rândul său, este mai lung decât segmentul PQ. Datorită faptului că această latură are ca extremități două vârfuri ale înfăşurătorii convexe, deducem imediat că punctele P şi Q nu se pot afla la distanța maximă.

În concluzie, primul pas necesar pentru a determina punctele aflate la distanța maximă este determinarea unei înfăşurători convexe.

O variantă care poate fi utilizată în continuare este considerarea succesivă a vârfurilor înfăşurătorii şi determinarea, printr-o metodă similară cu căutarea binară, a celui mai îndepărtat vârf față de vârful considerat.

Se va considera nodul din stânga (sau dreapta) nodului ales. Atâta timp cât distanța creşte, la fiecare pas k, vom "sări" la al 2^k-lea nod. În momentul în care distanța începe să scadă, vom reveni la nodul anterior şi vom începe să înjumătățim "distanța" saltului.

Dacă am ajuns la pasul l, acum, la fiecare, pas vom descreşte cu 1 puterea. Dacă la un anumit pas, puterea este p, vom efectua un salt la al 2^p-lea nod. Dacă distanța descreşte vom reveni, iar dacă nu, vom continua de la vârful respectiv. În final, vom obține vârful aflat la distanța maximă față de nodul considerat.

După considerarea tuturor vârfurilor, vom obține distanța maximă între două puncte din mulțimea dată.

Algoritmul descris funcționează corect deoarece, parcurgând nodurile într-un anumit sens distanța începe să crească până la un moment dat, după care ea începe să scadă.

O a doua variantă constă în alegerea unui vârf P_1 al înfăşurătorii şi parcurgerea acesteia (începând cu punctul din stânga sau din dreapta sa) atâta timp cât distanța creşte. Vom nota punctul respectiv prin P. În continuare vom trece la punctul P_2 (cel aflat pe înfăşurătoare imediat lângă punctul P_1, în funcție de sensul ales) şi vom continua parcurgerea. Poate fi uşor observat faptul că nu are sens să mai verificăm toate punctele parcurse la pasul anterior (cele până la P) deoarece distanțele obținute vor fi mai mici. Aşadar, vom continua de la punctul P atâta timp cât distanța va creşte. Vom trece apoi la punctul P_3 şi vom repeta în continuare procedeul până în momentul în care am luat în considerare toate punctele de pe înfăşurătoare. Evident, în final vom cunoaşte distanța maximă dintre două puncte de pe înfăşurătoare şi, implicit, distanța maximă dintre două puncte ale mulțimii date.

19.3.3. Analiza complexității

Aşa cum am arătat în capitolul 18, determinarea înfăşurătorii convexe are ordinul de complexitate $O(N \cdot \log N)$, dacă utilizăm scanarea *Graham* sau $O(N \cdot h)$, dacă utilizăm potrivirea *Jarvis*.

În situația în care, după determinarea înfăşurătorii, vom folosi metoda de căutare binară, distanța maximă va fi determinată într-un timp de ordinul $O(h \cdot \log h)$, datorită faptului că vom efectua h căutări binare, fiecare având ordinul de complexitate $O(\log h)$.

În cazul în care vom utiliza cea de-a doua metodă, aceasta va avea ordinul de complexitate $O(h)$, deoarece se poate observa faptul că prin avansări se va parcurge de cel mult două ori înfăşurătoarea.

Ca urmare, ordinul de complexitate al algoritmului de determinare a unei perechi de puncte aflate la distanță maximă este acelaşi cu cel al algoritmului folosit pentru determinarea înfăşurătorii convexe.

19.4. Rezumat

În cadrul acestui capitol am definit distanța euclidiană şi distanța Manhattan, după care am prezentat modul în care pot fi determinate, cea mai apropiată, respectiv cea mai îndepărtată, pereche de puncte dintr-o mulțime dată.

Pentru a determina cea mai apropiată pereche am prezentat un algoritm simplu, dar ineficient şi un algoritm eficient, bazat pe metoda *divide et impera*.

Pentru a determina cea mai îndepărtată pereche am prezentat un algoritm simplu, ineficient, foarte asemănător cu cel folosit pentru determinarea celei mai apropiate perechi de puncte, precum şi doi algoritmi bazați pe determinarea înfăşurătorii convexe a mulțimii.

19.5. Implementări sugerate

Dacă doriți să vă însuşiți mai bine cunoştințele referitoare la modul de determinare a distanței minime sau maxime între două puncte dintr-o mulțime, vă sugerăm să implementați algoritmi pentru:

1. determinarea distanței minime între două puncte considerând toate perechile posibile;
2. determinarea distanței maxime între două puncte considerând toate perechile posibile;
3. determinarea distanței minime între două puncte prin metoda *divide et impera*;
4. determinarea distanței maxime între două puncte prin determinarea unei înfăşurători convexe şi utilizarea unui număr de căutări liniare egal cu numărul vârfurilor înfăşurătorii;

5. determinarea distanței maxime între două puncte prin determinarea unei înfășurători convexe şi utilizarea unui număr de căutări binare egal cu numărul vârfurilor înfăşurătorii;
6. determinarea distanței maxime între două puncte prin determinarea unei înfășurători convexe şi ulterior, a perechii de puncte în timp liniar, în funcție de numărul vârfurilor de pe înfăşurătoare.

19.6. Probleme propuse

În continuare vom prezenta enunțurile câtorva probleme pe care vi le propunem spre rezolvare. Toate aceste probleme pot fi rezolvate folosind informațiile prezentate în cadrul acestui capitol. Cunoştințele suplimentare necesare sunt minime.

19.6.1. Copaci

Descrierea problemei

Pe un teren va fi plantată o pădure formată din N copaci. Se cunosc coordonatele la care va fi plantat fiecare copac, iar dimensiunile copacilor sunt neglijabile.

Riscul ca un copac să nu se dezvolte normal este invers proporțional cu distanța față de cel mai apropiat copac. Viitorul pădurar doreşte să determine unul dintre copacii pentru care riscul de a nu se dezvolta normal este maxim. Cu alte cuvinte, se doreşte determinarea unui copac pentru care distanța față de cel mai apropiat copac este minimă.

Date de intrare

Prima linie a fişierului de intrare `COPACI.IN` conține numărul N al copacilor care vor fi plantați. Fiecare dintre următoarele N linii conține câte o pereche de numere, separate printr-un spațiu, reprezentând coordonatele la care va fi plantat un copac.

Date de ieşire

Fişierul de ieşire `COPACI.OUT` va conține o singură linie pe care se vor afla coordonatele unuia dintre copacii pentru care riscul de a nu se dezvolta normal este maxim.

Restricții şi precizări

- $1 \leq N \leq 5000$;
- nu pot exista doi copaci la aceleaşi coordonate;
- toate coordonatele sunt numere întregi cuprinse între 0 şi 1000;
- întotdeauna vor exista cel puțin doi copaci pentru care riscul este maxim; în fişierul de ieşire vor fi scrise doar coordonatele unuia dintre aceşti copaci.

Exemplu

COPACI.IN	COPACI.OUT
5	0 0
0 0	
0 2	
1 1	
2 0	
2 2	

Timp de execuție: 1 secundă/test

19.6.2. Ştrumfi

Descrierea problemei

La concursul de îndemânare organizat de *Mare Ştrumf* au participat toți cei *N* ştrumfi din sat. După terminarea concursului, *Mare Ştrumf* s-a gândit la o modalitate de premiere care să pună din nou la încercare îndemânarea ştrumfilor.

El a plantat $2 \cdot N$ ciupercuțe pe un teren şi le-a spus ştrumfilor să aleagă perechi de ciupercuțe. Premiul primit de un ştrumf va fi direct proporțional cu distanța dintre ciupercuțele alese. Evident, ştrumfii vor alege ciupercuțele în ordinea din clasamentul concursului şi fiecare ştrumf va dori să primească un premiu cât mai mare, deci va alege perechea de ciupercuțe aflate la distanță maximă.

Pentru fiecare dintre cele $2 \cdot N$ ciupercuțe se cunosc coordonatele la care a fost plantată, iar dimensiunile ciupercuțelor sunt neglijabile.

Date de intrare

Prima linie a fişierului de intrare **STRUMFI.IN** va conține numărul ştrumfilor care au participat la concurs. Fiecare dintre următoarele $2 \cdot N$ linii conține câte o pereche de numere, separate printr-un spațiu, reprezentând coordonatele la care va fi plantată o ciupercuță.

Date de ieşire

Fişierul de ieşire **STRUMFI.OUT** va conține *N* linii; pe fiecare dintre acestea se va afla un număr reprezentând distanța dintre ciupercuțele alese de un ştrumf, precum şi numerele de ordine ale ciupercuțelor alese. Distanțele vor fi scrise în ordinea clasamentului final. Cu alte cuvinte, pe prima linie se va afla cea mai mare distanță, pe a doua linie se va afla cea mai mare distanță după eliminarea celor două ciupercuțe care au fost alese pentru determinarea primei distanțe şi aşa mai departe. Numerele de pe o linie vor fi separate prin spații.

Restricții şi precizări
- $1 \leq N \leq 500$;
- nu pot exista două sau mai multe ciupercuțe la aceleaşi coordonate;
- toate coordonatele sunt numere întregi cuprinse între 0 şi 1000;
- dacă un ştrumf are mai multe posibilități de a alege ciupercuțele, atunci el poate opta pentru oricare dintre acestea;
- distanțele din fişierul de ieşire vor fi scrise cu două zecimale exacte.

Exemplu

STRUMFI.IN	**STRUMFI.OUT**
4	2.83 1 8
0 0	2.83 3 6
0 1	2.00 2 7
0 2	2.00 4 5
1 0	
1 2	
2 0	
2 1	
2 2	

Timp de execuție: 1 secundă/test

19.6.3. Sportivi

Descrierea problemei
Pe un teren se află N obiective ale căror coordonate se cunosc. Doi sportivi trebuie să aleagă câte două obiective şi să alerge pe distanța dintre acestea. Unul dintre ei este în formă maximă şi va dori să alerge cât mai mult. Celălalt, tocmai şi-a reluat antrenamentele după o accidentare şi va dori să alerge cât mai puțin.

Aşadar, primul sportiv va alege obiectivele aflate la cea mai mare distanță, iar cel de-al doilea va alege obiectivele aflate la cea mai mică distanță.

Va trebui să determinați raportul dintre distanța parcursă de al doilea sportiv şi distanța parcursă de primul sportiv.

Date de intrare
Prima linie a fişierului de intrare **SPORTIVI.IN** conține numărul N al obiectivelor, iar fiecare dintre următoarele N linii conține câte o pereche de numere, separate printr-un spațiu, reprezentând coordonatele la care se află un obiectiv.

Date de ieşire
Fişierul de ieşire **SPORTIVI.OUT** va conține o singură linie pe care se va afla raportul dintre distanța parcursă de al doilea sportiv şi distanța parcursă de primul sportiv.

Restricții şi precizări

- $1 \leq N \leq 5000$;
- nu pot exista două obiective la aceleaşi coordonate;
- toate coordonatele sunt numere întregi cuprinse între 0 şi 1000;
- raportul determinat va fi scris în fişierul de ieşire cu opt zecimale exacte.

Exemplu

SPORTIVI.IN	SPORTIVI.OUT
5 0 0 0 2 1 1 2 0 2 2	0.50000000

Timp de execuție: 1 secundă/test

19.7. Soluțiile problemelor

Vom prezenta acum soluțiile problemelor propuse în cadrul secțiunii precedente. Pentru fiecare dintre acestea va fi descrisă metoda de rezolvare şi va fi analizată complexitatea algoritmului prezentat.

19.7.1. Copaci

Problema se reduce la a determina o pereche de puncte aflate la distanță minimă. După determinarea perechii de puncte vom scrie în fişierul de ieşire coordonatele unuia dintre cele două puncte.

Pentru aceasta va trebui doar să utilizăm algoritmul de determinare a celei mai apropiate perechi de puncte.

Analiza complexității

Citirea datelor de intrare implică citirea celor N perechi de coordonate ale copacilor care vor fi plantați, aşadar ordinul de complexitate al operației de citire a datelor de intrare este **O(N)**.

Algoritmul de determinare a celei mai apropiate perechi de puncte are ordinul de complexitate **O(N · log N)**.

Operația de scriere a datelor în fişierul de ieşire are ordinul de complexitate **O(1)** deoarece vom scrie doar două numere.

În concluzie, ordinul de complexitate al algoritmului de rezolvare a acestei probleme este $O(N) + O(N \cdot \log N) + O(1) =$ **O(N · log N)**.

19.7.2. Ştrumfi

Problema se reduce la a determina succesiv, cea mai îndepărtată pereche de puncte. Vom efectua, în total, N astfel de determinări. După fiecare determinare vom elimina punctele care formează cea mai îndepărtată pereche şi vom efectua următoarea determinare pentru punctele rămase. Ne vom opri în momentul în care sunt determinate toate cele N distanțe.

Aşadar, pentru a rezolva această problemă va trebui să aplicăm de N ori algoritmul de determinare a celei mai îndepărtate perechi de puncte.

Analiza complexității

Citirea datelor de intrare implică citirea celor $2 \cdot N$ perechi de coordonate ale ciupercuțelor, aşadar ordinul de complexitate al operației de citire a datelor de intrare este **O(N)**.

În continuare vom aplica de N ori algoritmul de determinare a celei mai îndepărtate perechi de puncte. Dacă avem K puncte, atunci ordinul de complexitate al operației va fi $O(K \cdot \log K)$. La primul pas vom avea $2 \cdot N$ puncte, la al doilea pas vom avea $2 \cdot N - 2$ puncte etc. Ca urmare, ordinul de complexitate al operației de determinare a tuturor celor N distanțe este calculat pe baza formulei:

$$\sum_{i=1}^{N} O\left(2 \cdot i \cdot \log(2 \cdot i)\right) = 2 \cdot \sum_{i=1}^{N} O\left(i \cdot \log(2 \cdot i)\right) =$$

$$2 \cdot \sum_{i=1}^{N} O\left(i \cdot \log i + i \cdot \log 2\right) = 2 \cdot \sum_{i=1}^{N} O\left(i \cdot \log i\right) \leq$$

$$2 \cdot \sum_{i=1}^{N} O\left(N \cdot \log N\right) = 2 \cdot O(N^2 \cdot \log N) = O(N^2 \cdot \log N)$$

Am obținut astfel o margine superioară pentru ordinul de complexitate. Se poate demonstra matematic că acesta este ordinul de complexitate real, dar calculele matematice sunt complicate, motiv pentru care nu le vom prezenta aici.

Scrierea datelor de ieşire se realizează pe parcursul determinării lor, deci nu se consumă timp suplimentar pentru această operație.

În concluzie, ordinul de complexitate al algoritmului de rezolvare a acestei probleme este $O(N) + O(N^2 \cdot \log N) = \mathbf{O(N^2 \cdot \log N)}$.

19.7.3. Sportivi

Problema se reduce la a determina o pereche de puncte aflate la distanță minimă, precum şi o pereche de puncte aflate la distanța maximă. După determinarea celor două perechi vom calcula distanțele dintre punctele care le formează, precum şi raportul dintre cele două distanțe.

Pentru aceasta va trebui să utilizăm atât algoritmul de determinare a celei mai apropiate perechi de puncte, cât şi algoritmul de determinare a celei mai îndepărtate perechi de puncte.

Analiza complexității

Citirea datelor de intrare implică citirea celor N perechi de coordonate ale obiectivelor, aşadar ordinul de complexitate al operației de citire a datelor de intrare este **O(N)**.

Algoritmul de determinare a celei mai apropiate perechi de puncte are ordinul de complexitate **O(N · log N)**.

În cazul în care pentru determinarea înfăşurătorii convexe folosim scanarea *Graham*, algoritmul de determinare a celei mai îndepărtate perechi de puncte are acelaşi ordin de complexitate **O(N · log N)**.

Operația de scriere a datelor în fişierul de ieşire are ordinul de complexitate **O(1)**, deoarece vom scrie un singur număr.

În concluzie, ordinul de complexitate al algoritmului de rezolvare a acestei probleme este $O(N) + O(N \cdot \log N) + O(N \cdot \log N) + O(1) =$ **O(N · log N)**.