NP-completitudine

Capitolul 11

- **Clase de probleme**
- **Reductibilitate**
- **Probleme NP-complete**
- **Concluzii**
- **Rezumat**
- **Implementări sugerate**
- **Probleme propuse**
- **Soluțiile problemelor**

Din punct de vedere al timpului de execuție, algoritmii pot fi împărțiți în două categorii: algoritmi polinomiali şi algoritmi exponențiali.

În cazul algoritmilor polinomiali, pentru date intrare având cardinalitate n, ordinul de complexitate în cel mai defavorabil caz are forma $O(n^k)$, unde k este o constantă.

Se pune în mod firesc întrebarea dacă orice problemă poate fi rezolvată folosind algoritmi polinomiali. Evident, răspunsul este ***nu***. De exemplu, problema generării tuturor permutărilor unei mulțimi are ordinul de complexitate $O(n!)$, deci timpul de execuție variază după o funcție suprapolinomială. Mai mult, există anumite probleme care nu pot fi rezolvate deloc, indiferent de timpul de execuție pe care l-ar avea la dispoziție un anumit algoritm.

De obicei, pentru problemele care pot fi rezolvate în timp polinomial se folosesc denumirile de problemă *accesibilă*, sau *tractabilă* (engl. *tractable*), iar pentru cele care nu pot fi rezolvate în timp polinomial se folosesc denumirile de problemă *inaccesibilă*, sau *netractabilă* (engl. *intractable*).

În cadrul acestui capitol vom prezenta o clasă specială de probleme, şi anume, aşanumitele probleme *NP-complete*. Pentru aceasta vom defini clasa problemelor rezolvabile în timp polinomial, precum şi clasa problemelor nerezolvabile în timp polinomial. De asemenea, vom descrie un mecanism care ne permite să stabilim legăturile dintre anumite probleme folosind reducerile în timp polinomial. În final vom defini noțiunea de *NP-completitudine* şi vom prezenta câteva probleme despre care se cunoaşte faptul că sunt NP-complete.

11.1. Clase de probleme

În cadrul acestei secțiuni vom clasifica diferitele probleme în funcție de timpul de execuție necesar execuției unui algoritm pentru rezolvarea lor. Vom începe cu clasa problemelor rezolvabile în timp polinomial, vom prezenta apoi problemele nedeterminist polinomiale şi în final problemele NP-complete.

11.1.1. Clasa P

În general, problemele rezolvabile în timp polinomial sunt privite ca fiind accesibile, chiar dacă există şi argumente împotriva unei astfel de afirmaţii.

În primul rând, deşi ar putea părea rezonabil să considerăm că o problemă este inaccesibilă dacă are ordinul de complexitate $O(n^{2457})$, practic nu există nici o problemă al cărei timp de execuţie să depindă de o funcţie polinomială cu un grad atât de mare. Deşi există posibilitatea enunţării de probleme al căror timp de execuţie să fie polinomial, dar funcţia polinomială să aibă un grad relativ mare, în practică foarte rar întâlnim probleme în care exponentul care apare în cadrul funcţiei polinomiale este mai mare decât 5. Ca urmare, problemele polinomiale întâlnite în practică necesită un timp relativ mic de execuţie.

În al doilea rând, se poate arăta faptul că dacă datele de ieşire ale unui algoritm polinomial devin date de intrare pentru un alt algoritm polinomial, atunci algoritmul "global" are şi el un timp de execuţie polinomial.

De asemenea, în cazul în care un algoritm polinomial apelează de un număr constant de ori subrutine polinomiale, atunci algoritmul rezultat este şi el polinomial.

Aşadar, clasa de probleme *P* este definită ca fiind totalitatea problemelor rezolvabile în timp polinomial.

11.1.2. Clasa NP

În cadrul acestei secţiuni vom introduce clasa problemelor nedeterminist polinomiale. Clasa problemelor *NP* reprezintă totalitatea problemelor care pot fi verificate în timp polinomial. Mai exact, o problemă este *NP* (*nedeterminist polinomială*) în cazul în care există un algoritm polinomial cu ajutorul căruia se poate verifica faptul că ieşirea unui algoritm de rezolvare a unei astfel de probleme este corectă sau nu.

Cu alte cuvinte, pentru ca o problemă să fie *NP*, trebuie să existe un algoritm din clasa *P* care are ca intrare atât o intrare, cât şi o ieşire a problemei *NP*. Algoritmul respectiv, numit algoritm de verificare, va trebui să determine dacă ieşirea furnizată este corectă pentru intrarea dată şi, evident, această verificare trebuie să poată fi realizată în timp polinomial.

De exemplu, nu avem nici un algoritm polinomial pentru rezolvarea problemei determinării unui ciclu hamiltonian într-un graf. Cu toate acestea, dacă avem un graf şi un anumit ciclu în graful respectiv, este foarte uşor să verificăm dacă ciclul este sau nu hamiltonian. Această verificare se poate realiza în timp liniar, deşi deocamdată nu se cunoaşte nici un algoritm polinomial pentru rezolvarea problemei ciclului hamiltonian.

Din nou se pune oarecum firesc întrebarea dacă pentru o problemă verificabilă în timp polinomial poate sau nu fi găsit întotdeauna un algoritm polinomial pentru rezolvarea ei. Cu alte cuvinte întrebarea se reduce la: $P = NP$?

Aceasta reprezintă una dintre cele mai importante întrebări ale informaticii teoretice. Până în acest moment nu se cunoaşte răspunsul, chiar dacă foarte mulţi teoreticieni bănuiesc că cele două clase de probleme nu sunt identice. Totuşi, nimeni nu a reuşit să

demonstreze adevărul sau falsitatea afirmației $P = NP$. În acest moment sunt oferite premii de milioane de dolari pentru primul care va reuşi să găsească răspunsul corect la această întrebare.

Totuşi, este evident că clasa problemelor P este inclusă în clasa problemelor NP, deoarece orice problemă rezolvabilă în timp polinomial poate fi verificată în timp polinomial. În cazul extrem, algoritmul de verificare ar reface rezolvarea dacă nu se descoperă o soluție mai rapidă. Chiar şi în aceste condiții, algoritmul de verificare este polinomial, deoarece el nu poate fi mai lent decât algoritmul de rezolvare.

Deşi cunoştințele actuale referitoare la relațiile dintre clasele P şi NP sunt oarecum lacunare, cercetând teoria NP-completitudinii se observă că modalitatea de demonstrare a inaccesibilității unei probleme este, din punct de vedere practic, mult mai simplă decât am putea presupune în acest moment.

11.1.3. Clasa NP-C

Unul dintre motivele pentru care informaticienii cred că mulțimile P şi NP nu sunt identice este existența problemelor NP-complete.

Cea mai surprinzătoare proprietate a acestei clase de probleme o reprezintă faptul că dacă se demonstrează că o problemă din această clasă este rezolvabilă în timp polinomial, atunci toate problemele din clasa respectivă sunt rezolvabile în timp polinomial.

De asemenea, dacă se demonstrează că o problemă din această clasă nu poate fi rezolvată în timp polinomial, atunci nici una dintre problemele din clasa respectivă nu poate fi rezolvată în timp polinomial.

Din nefericire, cu toate că aceste două afirmații au fost demonstrate, nimeni nu a reuşit până acum să rezolve în timp polinomial o problemă NP-completă şi nimeni nu a reuşit să demonstreze că o anumită problemă, despre care se ştie că este NP-completă, nu poate fi rezolvată în timp polinomial.

11.2. Reductibilitate

Intuitiv, putem spune că o problemă poate fi redusă la o altă problemă dacă orice instanță a primeia poate fi reformulată ca o instanță a celei de-a doua. Cu alte cuvinte, putem transforma intrarea primei probleme astfel încât să devină intrare pentru cea de-a doua şi putem transforma ieşirea celei de-a doua astfel încât să devină ieşire pentru prima.

Aşadar, în cazul în care există o funcție de transformare f care respectă cerințele de mai sus, o problemă este reductibilă la o altă problemă.

În cazul în care această funcție este calculabilă în timp polinomial, atunci prima problemă este reductibilă în timp polinomial la cea de-a doua.

Funcția de transformare f poartă denumirea de *funcție de reducere*, iar algoritmul folosit pentru transformare (de fapt, pentru calcularea funcției f) poartă denumirea de *algoritm de reducere*.

11.2.1. Reduceri pentru problemele P

Reducerile în timp polinomial ne oferă posibilitatea de a arăta că o anumită problemă este rezolvabilă în timp polinomial. Practic, dacă o problemă care face parte din clasa *P* este reductibilă în timp polinomial la o altă problemă, atunci această a doua problemă face cu siguranță parte din clasa *P*.

Trebuie observat faptul că dacă funcția de reducere nu poate fi evaluată în timp polinomial, atunci nu putem afirma cu siguranță că cea de-a doua problemă face parte din clasa *P*.

11.2.2. Definiția NP-Completitudinii

Practic, cu ajutorul reducerilor în timp polinomial putem stabili că o problemă este cel puțin la fel de dificil de rezolvat ca o altă problemă. Timpii de rezolvare necesari pentru cele două probleme vor diferi cel mult printr-un factor polinomial.

În acest moment putem defini clasa problemelor NP-C ca fiind formată din acele probleme ale clasei NP la care se pot reduce în timp polinomial toate problemele din clasa NP.

În cazul în care orice problemă NP poate fi redusă în timp polinomial la o anumită problemă, dar nu se ştie dacă această a doua problemă face parte din NP, atunci se spune că aceasta este NP-*dificilă* (engl. NP-*hard*).

11.3. Probleme NP-complete

După cum am arătat anterior, pentru a identifica problemele NP-complete folosind reducerile în timp polinomial, este necesar să ştim cu siguranță că există cel puțin o problemă care face parte din clasa NP-C.

Deşi ar putea părea oarecum bizar, demonstrarea NP-completitudinii unei probleme nu este suficient de simplă, precum ar părea. Prima problemă a cărei NP-completitudine a fost demonstrată este problema satisfiabilității formulei, al cărei enunț va fi prezentat în continuare.

Totuşi, această demonstrație este destul de greu de urmărit, motiv pentru care în ultima perioadă, în studiul NP-completitudinii se demonstrează mai întâi faptul că o altă problemă (numită problema satisfiabilității circuitului) este NP-completă. Chiar şi aşa, demonstrația este mult prea lungă şi necesită cunoştințe destul de avansate. Totuşi, concluzia este evidentă: există probleme NP-complete.

În cele ce urmează vom prezenta pe scurt enunțurile câtorva probleme a căror NP-completitudine a fost deja demonstrată. Bineînțeles, acestea reprezintă o foarte mică parte din totalul problemelor NP-complete, existând până acum sute de probleme a căror NP-completitudine a fost demonstrată. Probabil, cea mai interesantă dintre ele este problema jocului *Minesweeper*; aceasta s-a dovedit a fi NP-completă (fiind redusă la problema satisfiabilității formulei) la sfârşitul secolului al XX-lea.

11.3.1. Problema satisfiabilității circuitului

Un circuit combinațional logic este format din mai multe porți interconectate, fiecare dintre porți având una sau două intrări şi o singură ieşire. Intrările şi ieşirile sunt valori logice (0 sau 1).

Există trei tipuri de porți şi anume:

- porți ŞI: ieşirea este 1 dacă şi numai dacă cele două intrări ale porții au ambele valoarea 1; în caz contrar (cel puțin una dintre intrări are valoarea 0), ieşirea are valoarea 0;
- porți SAU: ieşirea este 1 dacă şi numai dacă cel puțin una dintre cele două intrări are valoarea 1; în caz contrar (ambele intrări au valoarea 0) ieşirea are valoarea 0;
- porți NU: ieşirea este 1 dacă şi numai dacă singura intrare are valoarea 0; în caz contrar (intrarea are valoarea 1) ieşirea este 0.

După interconectarea mai multor porți vom avea un număr de intrări (care nu sunt conectate cu ieşirea nici unei porți) şi o singură ieşire.

Problema satisfiabilității circuitului cere determinarea unui set de valori pentru intrări, astfel încât ieşirea circuitului să fie 1.

Aşa cum am afirmat anterior, această problemă este considerată a fi problema NP-completă "de bază". Ea va fi redusă în timp polinomial (de cele mai multe ori cu ajutorul unor probleme intermediare) la toate celelalte probleme NP-complete.

11.3.2. Problema satisfiabilității formulei

Cunoscută şi sub denumirea mai simplă de ***problema satisfiabilității***, această problemă are ca intrare o formulă logică ce conține operatori de disjuncție, conjuncție şi negație logică.

Se observă imediat corespondența dintre operatorii logici de la această problemă şi porțile logice de la problema precedentă. Demonstrația riguroasă este mai complicată, dar aceasta este ideea de bază. Practic, problema satisfiabilității circuitului poate fi redusă la problema satisfiabilității formulei, deci aceasta din urmă este şi ea NP-completă.

Această problemă poate fi enunțată astfel: se consideră o formulă logică formată din mai multe variabile logice şi mai mulți operatori logici; să se stabilească valorile care trebuie atribuite variabilelor pentru ca rezultatul evaluării expresiei să fie 1 (adevărat).

11.3.3. Problema satisfiabilității 3-FNC

Vom spune că o formulă logică se află în cea de-a *treia formă normală conjunctivă* (3-FNC) dacă este formată din mai multe conjuncții între disjuncții care conțin exact trei variabile sau negații ale acestora.

Problema satisfiabilității 3-FNC este foarte asemănătoare cu cea a satisfiabilității formulei, cerându-se practic tot o posibilitate de atribuire pentru valorile variabilelor. Singurul element suplimentar îl constituie restricția impusă asupra structurii formulei.

Se poate arăta faptul că pentru orice formulă logică poate fi identificată o formulă 3-FNC echivalentă, iar transformarea necesară se poate realiza în timp polinomial.

Ca urmare, folosind faptul că problema satisfiabilității formulei este *NP*-completă şi reducând-o în timp polinomial la problema satisfiabilității 3-FNC, deducem că şi aceasta din urmă este *NP*-completă.

11.3.4. Problema clicii

Vom numi *clică* un subgraf complet al unui graf. Problema clicii cere identificarea celei mai mari clici (care conține cele mai multe noduri) a unui graf dat.

Deşi ar putea părea ciudat, există o demonstrație riguroasă a faptului că problema satisfiabilității 3-FNC poate fi redusă în timp polinomial la problema clicii. Deoarece cunoaştem faptul că problema satisfiabilității 3-FNC este NP-completă, deducem că şi problema clicii este NP-completă.

Reiese acum motivul pentru care am prezentat şi problema satisfiabilitășii 3-FNC, cu toate că aceasta este de fapt doar un caz particular al problemei satisfiabilității formulei.

11.3.5. Problema acoperirii cu vârfuri

Vom defini o *acoperire cu vârfuri* a unui graf ca fiind o submulțime a nodurilor acestuia, astfel încât fiecare muchie a grafului să fie adiacentă cu cel puțin un nod din această submulțime. Cu alte cuvinte, cel puțin una dintre extremitățile fiecărei muchii trebuie să facă parte din acoperire.

Problema acoperirii cu vârfuri cere determinarea unei acoperiri care să conțină un număr minim de elemente.

Şi pentru această problemă avem o reducere în timp polinomial de la o problemă NP-completă şi anume, după cum probabil bănuiți, de la problema clicii. Ca urmare, problema acoperirii cu vârfuri este NP-completă.

11.3.6. Problema sumei submulțimii

Această problemă apare de foarte multe ori la concursurile de programare, sub diverse forme. Practic se consideră o mulțime cu mai multe elemente şi o valoare dată şi se cere determinarea unei submulțimi care să aibă suma elementelor egală cu valoarea dată.

Există mai multe abordări pentru rezolvarea acestei probleme, dar nu există nici o rezolvare în timp polinomial. De obicei, se foloseşte metoda programării dinamice care duce la obținerea unei soluții într-un timp polinomial ce depinde de valoarea căutată. Totuşi, valoarea căutată nu depinde de numărul elementelor mulțimii (adică de dimensiunea intrării), deci algoritmul nu are timp de execuție polinomial față de dimen-

siunea mulțimii. În această situație se spune, uneori, că timpul de execuție al algoritmului este pseudopolinomial.

După cum probabil bănuiți, este destul de puțin probabil să se descopere un algoritm care să ruleze în timp polinomial pentru cazul general, deoarece problema sumei submulțimii este NP-completă.

Este destul de surprinzător faptul că pentru demonstrarea NP-completitudinii acestei probleme s-a folosit o reducere în timp polinomial de la problema acoperii cu vârfuri. Deşi puțini s-ar fi aşteptat la aşa ceva, o astfel de demonstrație există, deci problema sumei submulțimii este NP-completă.

11.3.7. Problema ciclului hamiltonian

Această problemă nu mai are nevoie de nici o prezentare. Enunțul este foarte simplu: dându-se un graf neorientat, să se identifice un ciclu elementar care conține toate nodurile grafului.

La fel ca şi pentru problema clicii, demonstrarea NP-completitudinii problemei ciclului hamiltonian se bazează pe o reducere în timp polinomial de la problema satisfiabilității 3-FNC.

Observăm acum că această din urmă problemă pare a fi mult mai importantă decât în momentul în care am amintit-o pentru prima dată. De fapt, ea este utilizată foarte des pentru demonstrarea NP-completitudinii unor probleme care par a nu avea nici o legătură cu formulele logice.

11.3.8. Problema comis-voiajorului

O problemă foarte asemănătoare cu cea a ciclului hamiltonian este cea a comis-voiajorului. Practic, pentru această problemă avem costuri ataşate muchiilor grafului şi dorim să determinăm un ciclu hamiltonian în care costul total al muchiilor să fie cât mai mic posibil.

Este evident faptul că, dacă pentru toate muchiile avem costul 1, obținem chiar problema ciclului hamiltonian. Pe baza acestei observații este relativ uşor să reducem în timp polinomial problema ciclului hamiltonian la cea a comis-voiajorului. Ca urmare, problema comis-voiajorului este NP-completă.

11.3.9. Problema mulțimii independente

Această problemă este foarte asemănătoare cu problema acoperirii cu vârfuri. Ea cere determinarea unei submulțimi a vârfurilor astfel încât fiecare muchie a grafului să fie adiacentă cu cel mult un nod din această submulțime. Evident, vom dori să obținem o submulțime care să conțină cât mai multe elemente.

Pentru demonstrarea NP-completitudinii acestei probleme s-a realizat o reducere în timp polinomial de la problema clicii.

11.3.10. Problema colorării

Şi această problemă este destul de cunoscută: se consideră un graf şi se cere atribuirea unor numere (culori) pentru vârfurile sale, astfel încât să nu existe două vârfuri adiacente colorate cu aceeaşi culoare. Problema cere, de asemenea, ca numărul valorilor distincte (numărul culorilor) să fie minim.

Demonstraţia NP-completitudinii acestei probleme s-a realizat în doi paşi. În primul rând s-a demonstrat faptul că problema colorării cu trei culori poate fi redusă în timp polinomial la problema generală a colorării.

Problema colorării cu trei culori este foarte asemănătoare cu problema generală a colorării (de fapt este echivalentă, deoarece ambele sunt NP-complete, după cum vom vedea imediat) şi cere colorarea vârfurilor folosind cel mult trei culori, dacă este posibil.

După demonstrarea acestei reduceri, urmează să arătăm că problema colorării cu trei culori este NP-completă şi vom deduce imediat că şi problema generală a colorării este NP-completă.

După cum probabil bănuiţi (datorită apariţiei numărului trei) există posibilitatea de a reduce în timp polinomial problema satisfiabilităţii 3-FNC la problema colorării cu trei culori. Ca urmare, deducem că în cazul în care avem cel puţin trei culori, problema colorării este NP-completă.

Expresia "cel puţin" din paragraful anterior are o importanţă destul de mare, deoarece există algoritmi polinomiali pentru realizarea colorărilor cu două culori, dacă astfel de colorări există.

11.3.11. Problema jocului Minesweeper

Aşa cum am afirmat anterior, chiar şi problema acestui celebru joc este *NP*-completă. Evident, enunţul acesteia nu poate fi cerinţa de a juca propriu-zis Minesweeper.

Datorită faptului că jocul este foarte cunoscut, nu vom aminti acum regulile acestuia, ci vom prezenta doar enunţul problemei: dându-se o configuraţie parţială a tablei (cu anumite căsuţe acoperite) să se decidă dacă există sau nu o singură posibilitate pentru valorile căsuţelor acoperite.

Cu alte cuvinte, se cere verificarea faptului că jucătorul poate continua fără a recurge la noroc pentru a descoperi toate minele...

Din nou surpriza vine de la problema care a fost redusă în timp polinomial la problema jocului Minesweeper. Dar, după cum probabil v-aţi obişnuit deja, este vorba de problema satisfiabilităţii formulei 3-FNC. Nu are nici un rost să vă îndoiţi de această afirmaţie... Vă asigurăm că demonstraţia există şi a fost acceptată de comunitatea ştiinţifică.

11.4. Concluzii

Din acest capitol reiese destul de clar faptul că este foarte puțin probabil să găsiți un algoritm rapid pentru o problemă NP-completă. Recunoaşterea NP-completitudinii unor probleme duce la o economie destul de importantă de timp, deoarece veți şti că nu are sens să încercați să găsiți un algoritm rapid. Nimeni nu a reuşit până acum să descopere un astfel de algoritm.

Aşadar, în loc să încercați să găsiți algoritmi polinomiali pe care aveți foarte puține şanse să îi descoperiți, vă veți putea concentra imediat asupra altor metode de rezolvare, cum ar fi cele care vor fi descrise în capitolele următoare.

11.5. Rezumat

În cadrul acestui capitol am introdus noțiunea de NP-completitudine, am prezentat pe scurt modul în care poate fi demonstrată sau verificată NP-completitudinea unor probleme şi am prezentat mai multe exemple de probleme NP-complete mai cunoscute.

De asemenea, am definit clasele de probleme P, NP şi NP-C şi am concluzionat că există foarte puține şanse de a se descoperi algoritmi rapizi pentru rezolvarea problemelor NP-complete şi este recomandabil ca, după ce se dovedeşte că o problemă este NP-completă, să se încerce alte abordări pentru rezolvarea acesteia.

11.6. Implementări sugerate

Pentru a observa timpul lent de execuție pe care îl au algoritmii de rezolvare a problemelor NP-complete (în cazurile generale) vă sugerăm să încercați să implementați algoritmi cât mai rapizi pentru:

1. problema satisfiabilității formulei;
2. problema clicii;
3. problema acoperirii cu vârfuri;
4. problema sumei submulțimii;
5. problema ciclului hamiltonian;
6. problema comis-voiajorului;
7. problema mulțimii independente;
8. problema colorării.

11.7. Probleme propuse

În continuare vom prezenta enunțurile câtorva probleme pe care vi le propunem spre rezolvare. Toate acestea sunt *NP*-complete, dar dimensiunile datelor de intrare sunt mici, motiv pentru care pot fi rezolvate folosind algoritmi exponențiali.

11.7.1. Ambasada

Ambasada marțiană pe Pământ organizează o mică recepție la care sunt invitați ambasadori de pe toate planetele locuite şi de pe toți sateliții locuiți din *Sistemul Solar*.

Fiecare ambasador cunoaşte un număr de limbi străine şi, evident, doi ambasadori vor putea discuta fără a avea nevoie de traducător, dacă există cel puțin o limbă pe care o vorbesc şi o înțeleg amândoi.

Gazda recepției doreşte să determine cel mai mare grup de ambasadori, astfel încât fiecare membru al grupului poate discuta direct cu oricare alt membru al grupului.

Date de intrare

Prima linie a fişierului de intrare **AMBASADA.IN** conține numărul N al ambasadorilor prezenți la recepție. Pe fiecare dintre următoarele N linii sunt prezentate limbile străine corespunzătoare unui ambasador. Primul număr de pe o astfel de linie este numărul l al limbilor vorbite de ambasador, iar următoarele l numere reprezintă numerele de ordine ale celor l limbi. Aceste l numere vor fi distincte, iar numerele de pe o linie vor fi separate prin spații.

Date de ieşire

Fişierul de ieşire **AMBASADA.OUT** va conține două linii. Pe prima dintre acestea va fi scris numărul ambasadorilor care fac parte din grupul determinat, iar pe cea de-a doua linie vor fi scrise numerele de ordine ale acestor ambasadori. Numerele de pe această linie vor fi separate prin spații.

Restricții şi precizări

- $2 \leq N \leq 16$;
- ambasadorii sunt identificați prin numere cuprinse între 1 şi N;
- un ambasador poate cunoaşte cel mult zece limbi străine;
- dacă există mai multe soluții, trebuie determinată doar una dintre ele.

Exemplu

AMBASADA.IN	AMBASADA.OUT
5 3 1 2 5 2 3 4 4 1 2 3 4 2 4 5 2 2 3	3 2 3 5

Timp de execuție: 1 secundă/test

11.7.2. Turism

O agenție de turism dorește să organizeze o excursie în care turiștii să viziteze N orașe pornind dintr-un oraș dat. Pentru fiecare pereche de orașe este cunoscut costul călătoriei între cele două orașe. Evident, se dorește să se aleagă un circuit care trece o singură dată prin fiecare oraș (cu excepția celui din care se pornește) astfel încât costul total al călătoriilor să fie minim.

Date de intrare

Prima linie a fișierului de intrare **TURISM.IN** conține numărul N al orașelor care trebuie vizitate. Pe fiecare dintre următoarele N linii sunt prezentate costurile călătoriilor între orașe sub forma unei matrice A cu N linii și N coloane. Elementele unei linii a matricei vor apărea pe aceeași linie a fișierului și vor fi separate prin spații. Liniile și coloanele vor fi ordonate în funcție de numerele de ordine ale orașelor. Cu alte cuvinte, prima dintre aceste linii va conține costurile călătoriilor pornind de la primul oraș, cea de-a doua linie va conține costurile călătoriilor pornind de la al doilea oraș. De asemenea, prima coloană va conține costurile călătoriilor până la primul oraș, cea de-a doua coloană va conține costurile călătoriilor până la al doilea oraș etc.

Date de ieșire

Fișierul de ieșire **TURISM.OUT** va conține o singură linie pe care se vor afla $N - 1$ numere întregi care reprezintă numerele de ordine ale orașelor în ordinea în care acestea vor fi parcurse. Orașul din care se pleacă nu va apărea pe această linie.

Restricții și precizări

- $2 \leq N \leq 10$;
- orașele sunt identificate prin numere cuprinse între 1 și N;
- se va pleca întotdeauna din orașul identificat prin numărul 1;
- costul unei călătorii este un număr întreg cuprins între 1 și 1000;
- elementele de pe diagonala principală a matricei A vor avea întotdeauna valoarea 0;
- un element A_{ij} va avea întotdeauna valoarea egală cu cea a elementului A_{ji};
- dacă există mai multe soluții, trebuie determinată doar una dintre ele.

Exemplu

TURISM.IN	TURISM.OUT
4 0 1 2 3 1 0 4 5 2 4 0 6 3 5 6 0	2 3 4

Timp de execuție: 1 secundă/test

11.7.3. Bancnote

Un cumpărător are la dispoziție mai multe bancnote cu valori nu neapărat distincte. El trebuie să plătească o sumă *S*, folosind doar bancnotele pe care le are la dispoziție. În cazul în care nu poate obține suma *S*, va trebui să obțină o sumă mai mare, dar foarte apropiată de *S*. Va trebui să determinați care este cea mai mică sumă mai mare sau egală cu *S* care poate fi obținută.

Date de intrare

Prima linie a fișierului de intrare **BANCNOTE.IN** conține numărul *N* al bancnotelor pe care le are la dispoziție cumpărătorul, precum şi suma *S* care trebuie plătită. Cea de-a doua linie a fișierului va conține *N* numere întregi, separate prin spații, care reprezintă valorile celor *N* bancnote.

Date de ieşire

Fişierul de ieşire **BANCNOTE.OUT** va conține o singură linie pe care se va afla un singur număr reprezentând cea mai mică sumă mai mare sau egală cu *S* care poate fi obținută.

Restricții şi precizări

- $1 \leq N \leq 100$;
- $1 \leq S \leq 10000$;
- valoarea unei bancnote este un număr întreg cuprins între 1 şi 10000;
- va exista întotdeauna posibilitatea de a plăti o sumă cel mult egală cu dublul valorii *S*.

Exemplu

BANCNOTE.IN
```
10 2457
1000 1000 500 200 200 100 50 20 20 10
```

BANCNOTE.OUT
```
2460
```

Timp de execuție: 1 secundă/test

11.8. 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.

11.8.1. Ambasada

Vom construi un graf neorientat ale cărui noduri vor reprezenta ambasadorii. Va exista o muchie între două noduri dacă există cel puțin o limbă care este vorbită de către ambii ambasadori care corespund celor două noduri.

Astfel, problema se reduce la determinarea unei clici maximale într-un graf neorientat. Deşi, problema clicii este NP-completă, numărul mic al ambasadorilor ne permite să implementăm un program care să o rezolve în toate situațiile posibile.

Vom genera toate cele 2^N submulțimi ale mulțimii nodurilor şi vom verifica, pentru fiecare mulțime, dacă aceasta reprezintă o clică. Vom păstra elementele celei mai mari clici obținute şi, în final, vom scrie în fişierul de ieşire dimensiunea clicii, precum şi numerele de ordine ale nodurilor care o formează.

Analiza complexității

Operația de citire a datelor de intrare are ordinul de complexitate **O(N)** deoarece se citesc date referitoare la N ambasadori. Considerăm că citirea datelor referitoare la un ambasador se realizează în timp constant, deoarece acesta poate vorbi cel mult zece limbi.

După citirea datelor va trebui să creăm graful care va conține cel mult N^2 muchii. Verificarea existenței unei muchii între două noduri se realizează în timp constant, deoarece va trebui să efectuăm cel mult o sută de comparații. Aşadar, operația de creare a grafului bipartit va avea ordinul de complexitate $\mathbf{O(N^2)}$.

În continuare va trebui să determinăm toate submulțimile unei mulțimi cu N elemente; numărul acestora este 2^N. Pentru fiecare submulțime va trebui să verificăm dacă aceasta reprezintă o clică, operație ce implică verificarea existenței unei muchii între oricare pereche de noduri. Datorită faptului că putem avea cel mult N noduri, ordinul de complexitate al acestei operații va fi $O(N^2)$. Aşadar, pentru a determina clica maximă folosind această metodă avem nevoie de un timp de ordinul $\mathbf{O(2^N \cdot N^2)}$.

Afişarea soluției constă în scrierea celor cel mult N elemente ale unei clici, deci ordinul de complexitate al operației este $O(N)$.

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

11.8.2. Turism

Vom privi cele N oraşe ca fiind nodurile unui graf complet ale căror muchii au asociate costuri. Evident, costul unei muchii va fi egal cu costul călătoriei între cele două oraşe care reprezintă extremitățile muchiei.

Aşadar, am redus problema la cunoscuta problemă NP-completă a comis-voiajorului. Din fericire, vom avea cel mult 10 oraşe, motiv pentru care putem să încercăm toate traseele posibile şi să îl alegem pe cel mai bun.

Datorită faptului că există drumuri între oricare două oraşe, orice permutare a mulțimii {1, …, *N*} este un posibil traseu. Datorită faptului că pornim întotdeauna din oraşul identificat prin 1, este suficient să determinăm permutările mulțimii {2, …, *N*}.

Pentru fiecare permutare vom calcula costul traseului corespunzător şi vom păstra traseul cu cel mai mic cost. Acest traseu va fi descris în fişierul de ieşire.

Analiza complexității

Operația de citire a datelor de intrare are ordinul de complexitate $\mathbf{O(N^2)}$, deoarece se citeşte întreaga matrice a costurilor pentru graful construit.

În continuare vom genera toate cele $(N - 1)!$ permutări ale mulțimii {2, …, *N*}. Pentru fiecare permutare vom calcula costul traseului corespunzător, operație ce se realizează în timp liniar. Ca urmare, ordinul de complexitate al algoritmului de determinare a traseului de cost minim va fi $O((N - 1)!) \cdot O(N) = \mathbf{O(N!)}$.

Pentru afişarea soluției, va trebui să parcurgem traseul determinat, deci această operație are ordinul de complexitate **O(N)**.

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

11.8.3. Bancnote

Imediat după citirea enunțului observăm faptul că aceasta este o variantă a problemei submulțimii de sumă dată.

Din fericire, chiar dacă pentru cazul general această problemă este NP-completă, particularitățile permit găsirea unui algoritm cu timp de execuție pseudopolinomial.

Vom păstra un şir de valori booleene ale cărui elemente sunt inițializate cu valoarea *fals*, cu excepția elementului cu indicele 0. Semnificația şirului este următoarea: dacă al *i*-lea element are valoarea *adevărat*, atunci suma *i* poate fi plătită folosindu-se bancnotele date.

La fiecare pas, vom verifica care sunt sumele care pot fi plătite în urma paşilor anteriori; dacă valoarea bancnotei curente este *b* şi suma *s* a putut fi plătită în urma paşilor anteriori, atunci la pasul curent poate fi plătită suma $b + s$.

În final vor fi marcate toate sumele care pot fi plătite folosind bancnotele pe care le are la dispoziție cumpărătorul. Va fi suficient să o alegem pe cea mai mică care este mai mare sau egală cu *S*. Pentru aceasta vom parcurge elementele şirului, începând cu poziția *S* şi ne vom opri în momentul în care valoarea elementului curent este *adevărată*.

Există o altă particularitate a problemei care limitează dimensiunea şirului utilizat. Evident, nu ne interesează dacă putem plăti sume foarte mari. Cum ştim că vom putea întotdeauna plăti o sumă cel mult egală cu $2 \cdot S$, putem ignora toate valorile mai mari decât $2 \cdot S$. Aşadar, vom lucra cu un şir de valori booleene care are $2 \cdot S$ elemente.

Analiza complexității

Operația de citire a datelor de intrare are ordinul de complexitate **O(N)**, deoarece se citesc valorile celor N bancnote.

În continuare la fiecare pas va trebui să parcurgem şirul valorilor booleene; datorită faptului că acesta are $2 \cdot S$ elemente, ordinul de complexitate al unei parcurgeri va fi $O(2 \cdot S) = O(S)$. Aşadar, operația de determinare a valorilor şirului are ordinul de complexitate **O(N · S)**.

Pentru afişarea soluției va trebui să parcurgem şirul începând cu poziția S. Datorită faptului că există $S + 1$ elemente începând cu această poziție, ordinul de complexitate al acestei operații este **O(S)**.

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