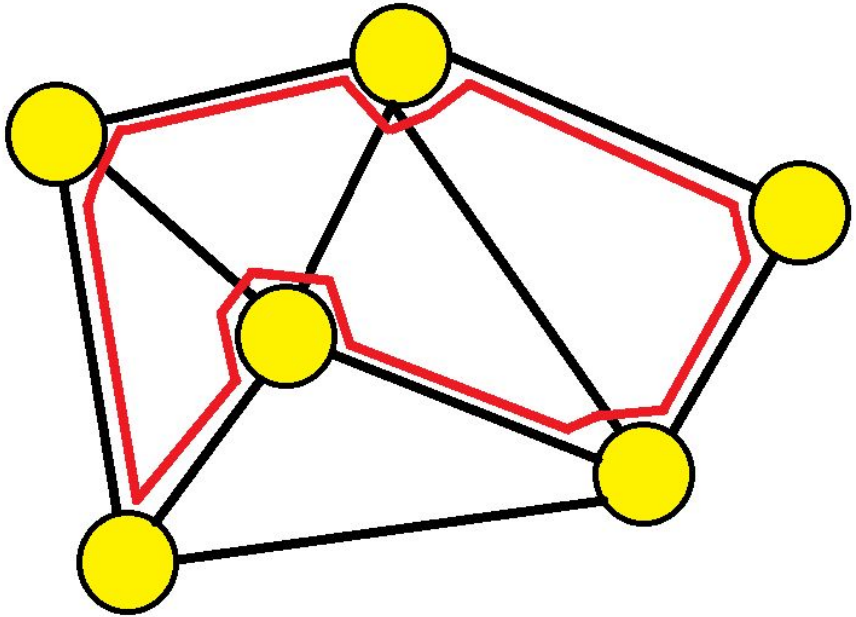
P VS NP & Algoritmi Aproximativi

Mare parte din curs se bazează pe Algoritmi

Recapitulare discutii Hamiltonietate

Un graf este hamiltonian dacă avem un ciclu care trece prin toate nodurile exact o data



Recapitulare discutii Hamiltonietate

- **Nu este un cunoscut un algoritm polinomial pentru a găsi un ciclu hamiltonian.**

Recapitulare discutii Hamiltonietate

- **Nu este un cunoscut un algoritm polinomial pentru a găsi un ciclu hamiltonian.**
  - Exista **algoritmi exponentiali** pentru a găsi un ciclu hamiltonian O(n!) și O(2^n*n) ba chiar în O(1.657^n).

Recapitulare discutii Hamiltonietate

- **Nu este un cunoscut un algoritm polinomial pentru a găsi un ciclu hamiltonian.**
  - Exista **algoritmi exponentiali** pentru a găsi un ciclu hamiltonian O(n!) și O(2^n*n) ba chiar în O(1.657^n).
  - Exista un **algoritm liniar (polinomial)** de a determina dacă un sir de numere reprezinta un ciclu hamiltonian într-un graf.

Recapitulare discutii Hamiltonietate

- **Nu este un cunoscut un algoritm polinomial pentru a găsi un ciclu hamiltonian.**
  - Exista **algoritmi exponentiali** pentru a găsi un ciclu hamiltonian O(n!) și O(2^n*n) ba chiar în O(1.657^n).
  - Exista un **algoritm liniar (polinomial)** de a determina dacă un sir de numere reprezinta un ciclu hamiltonian într-un graf.
    - Vom spune ca acest algoritm este un **oracol** (care poate verifica în timp polinomial o soluție).

Recapitulare discutii Hamiltonietate

- **Nu este un cunoscut un algoritm polinomial pentru a găsi un ciclu hamiltonian.**
  - Exista **algoritmi exponentiali** pentru a găsi un ciclu hamiltonian O(n!) și O(2^n*n) ba chiar în O(1.657^n).
  - Exista un **algoritm liniar (polinomial)** de a determina dacă un sir de numere reprezinta un ciclu hamiltonian într-un graf.
    - Vom spune ca acest algoritm este un **oracol** (care poate verifica în timp polinomial o soluție).
  - Exista algoritm liniar nedeterminist pentru a determina dacă un graf este hamiltonian.

Complexitatea unei probleme

Dacă avem un algoritm exponențial pentru o problema înseamnă ca problema are complexitate exponențială ?

Complexitatea unei probleme

Dacă avem un algoritm exponențial pentru o problema înseamnă ca problema are complexitate exponențială ?

- Nu, poate exista și alți algoritmi mai eficienti….
- De exemplu algoritmul Bogosort are expected running time în O((n+1)!), asta nu înseamnă ca sortarea este o problema exponențială.
    - De fapt chiar știm ca pentru un vector arbitrar de n elemente putem rezolva problema în complexitate ???

Complexitatea unei probleme

Dacă avem un algoritm exponențial pentru o problema înseamnă ca problema are complexitate exponențială ?

- Nu, poate exista și alți algoritmi mai eficienti….
- De exemplu algoritmul Bogosort are expected running time în O((n+1)!), asta nu înseamnă ca sortarea este o problema exponențială.
  - De fapt chiar știm ca pentru un vector arbitrar de n elemente putem rezolva problema în complexitate O(n log n)

Complexitatea unei probleme

- În general pentru o problema vom avea:
  - Limita teoretica inferioara (pentru sortare e O(n log n))
  - Complexitatea problemei (pentru sortare e O(n log n))
  - Algoritm optim cunoscut (pentru sortare e tot O(n logn))

Complexitatea unei probleme

- În general pentru o problema P vom avea
  - Limita teoretica inferioara (pentru sortare e O(n log n))
  - Complexitatea problemei (pentru sortare e O(n log n))
  - Algoritm optim cunoscut (pentru sortare e tot O(n logn))
- Înmulțirea a două numere (numerele scrise în baza 2, iar mărimea lor numărul total de biţi n):
  - Limita teoretica inferioara (O(n))
  - Complexitatea problemei (????)
  - Algoritm optim cunoscut (O(n log n log log n))

Complexitatea unei probleme

- În general pentru o problema P vom avea
  - Limita teoretica inferioara (pentru sortare e O(n log n))
  - Complexitatea problemei (pentru sortare e O(n log n))
  - Algoritm optim cunoscut (pentru sortare e tot O(n logn))
- Înmulțirea a două numere (numerele scrise în baza 2, iar mărimea lor numărul total de biţi n):
  - Limita teoretica inferioara (O(n))
  - Complexitatea problemei (????)
  - Algoritm optim cunoscut (O(n log n log log n))
- Ciclu hamiltonian
  - Limita teoretica inferioara (O(n)) -> polinomial
  - Complexitatea problemei (????)
  - Algoritm optim cunoscut (O(1.657^n)) -> exponential

P vs NP

Complexity zoo

Ce înseamnă ca o problema e NP ?

Ce înseamnă NP ?

P

P este clasa problemelor de decizie (probleme care răspund la întrebări cu da și nu) ce se pot rezolva în timp polinomial:

- Este un element intr-un vector
  - Rezolvabila în timp liniar
- Are graful G un cuplaj de mărime k?
  - O(n^?? (2/3/4/5/6))

P

P este clasa problemelor de decizie (probleme care răspund la întrebări cu da și nu) ce se pot rezolva în timp polinomial:

- Este un element intr-un vector
  - Rezolvabila în timp liniar
- Are graful G un cuplaj de mărime k?
  - O(n^?? (2/3/4))
- Este numărul n prim ? problema rezolvata recent
  - De ce rezolvata recent ??

P

P este clasa problemelor de decizie (probleme care răspund la întrebări cu da și nu) ce se pot rezolva în timp liniar:

- Este un element intr-un vector
  - Rezolvabila în timp liniar
- Are graful G un cuplaj de mărime k?
  - O(n^?? (2/3/4))
- Este numărul n prim ? problema rezolvata recent
  - De ce rezolvata recent ??
  - Spunem n dar de fapt n este exponențial în numărul de cifre din input (n = 54354 = 10^4 *5 + ….). Un algoritm O(n/2) sau o(sqrt(n)) este de fapt exponențial în mărimea inputului…
  - Algoritm în O(logn^(15/2)) (algoritm din 2002)

NP

NP este clasa problemelor de decizie (probleme care răspund la întrebări cu da și nu) ce se pot rezolva în timp liniar pe o masina nedeterminista.

NP:

- **Nedeterminist Polinomial**
- ~~**Nepolinomial**~~

NP

Cilcul Hamiltonian:

```
A: For (i =1; i <= n; ++i) {
    If (nr == n && v[last][first]==1 ) return true;
    if(sel[i] == 0 && v[last][i] == 1)
        { sel[i] = 1; nr++; goto A; sel[i] = 0; nr--; last = i;}
} return false;
```

NP

Cilcul Hamiltonian:

```
A: For (i =1; i <= n; ++i) {
    If (nr == n && v[last][first]==1 ) return true;
    if(sel[i] == 0 && v[last][i] == 1)
        { sel[i] = 1; nr++; goto A; sel[i] = 0; nr--; last = i;}
} return false;
```

Complexitate ?

NP

Cilcul Hamiltonian:

```
A: For (i =1; i <= n; ++i) {
    If (nr == n && v[last][first]==1 ) return true;
    if(sel[i] == 0 && v[last][i] == 1)
        { sel[i] = 1; nr++; goto A; sel[i] = 0; nr--; last = i;}
} return false;
```

Complexitate polinomiala (dar nedeterminist)

NP

Nu toate problemele sunt în NP!

- **Probleme nedecidabile:**
  - **Cea mai cunoscută Halting problem prezentată de Alan Turing (Imitation game este un film despre el)**
  - **Halting problem:**
    - *Nu există o maşină Turing, care primind la intrare descrierea unei alte maşini Turing T şi un şir de date de intrare x, să poată spune dacă T se opreşte vreodată când primeşte pe x la intrare*
    - *Practic nu putem sti pentru orice program dacă se oprește pentru orice input…*
  - Teoria complexitatii de Mihai Budiu
- *Problema exponentiale…*
    - *Aici o sa mai vorbim un pic mai târziu, dar exista probleme mai incete decat NP*
  -

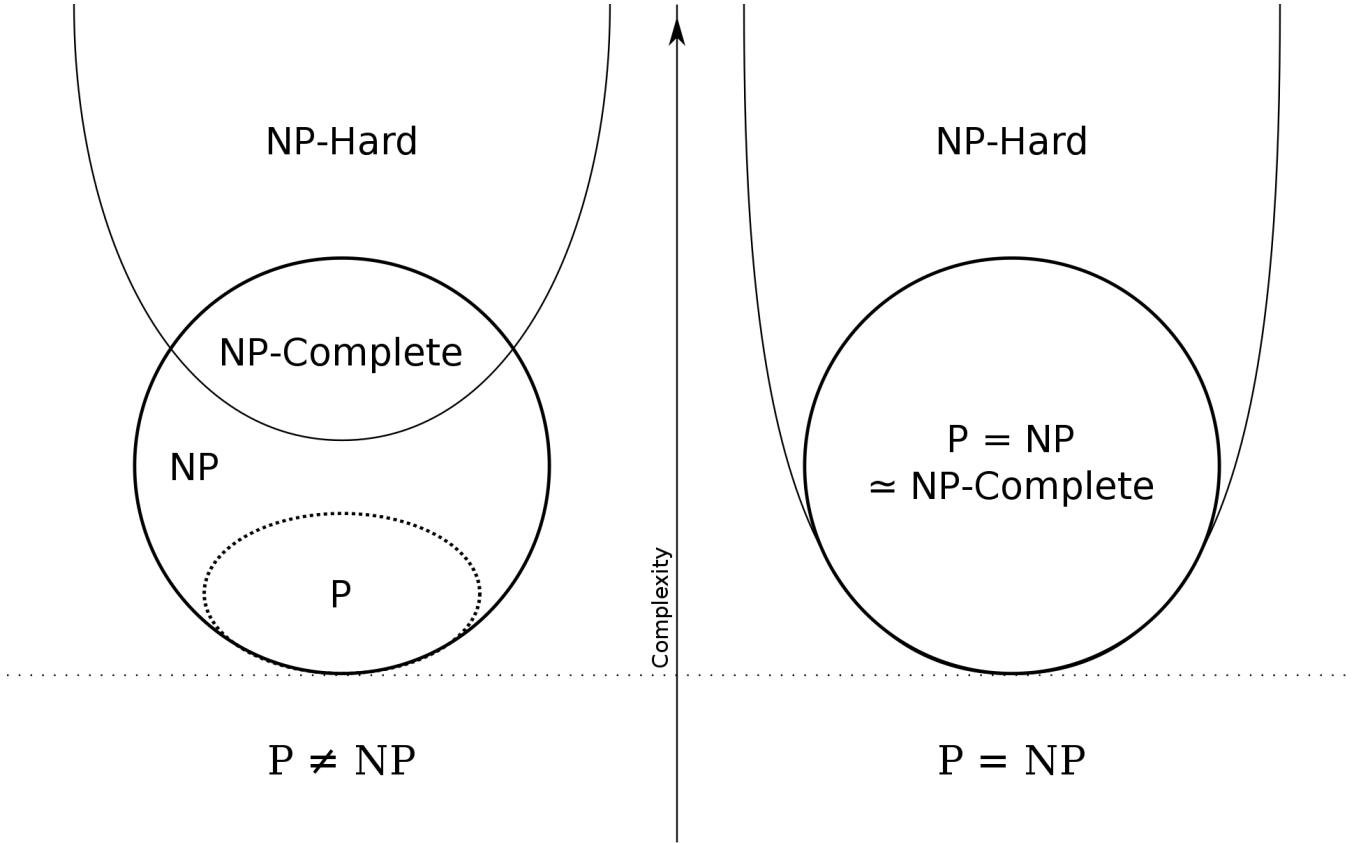
NP

Probleme NP complete:

- *Exista o lista de probleme care are proprietatea ca dacă se găsește rezolvarea pentru ele în timp polinomial toate problemele din NP se pot rezolva în timp polinomial*
  - *Ciclu Hamiltionian (Avem și film)*
  - Dându-se o mulţime de numere naturale, se poate împărţi în două mulţimi de numere de sume egale?
  - Clica": dându-se un graf G şi un număr k, are G un subgraf complet cu k vârfuri (adică o mulţime de k vârfuri unite fiecare cu fiecare)?
  - ``Acoperire": dându-se un graf G şi un număr k, pot alege k vârfuri în aşa fel încît toate muchiile din G au un capăt ales?

P VS NP

Wikipedia:



P VS NP VS EXP

Clasa tuturor problemelor care se pot rezolva cu algoritmi nedeterminişti într-un timp polinomial se notează cu **NP (Nedeterminist Polinomial**). Este clar că orice problemă care se află în P se află şi în NP, pentru că algoritmii determinişti sunt doar un caz extrem al celor determinişti: în fiecare moment au o singură alegere posibilă.

Din păcate transformarea într-un algoritm determinist se face pierzând din eficienţă. În general un algoritm care operează în timp nedeterminist polinomial (NP) poate fi transformat cu uşurinţă într-un algoritm care merge în timp exponenţial (EXP)**. Avem deci o incluziune de mulţimi între problemele de decizie: P $\subseteq$ NP $\subseteq$ EXP.**

Partea cea mai interesantă este următoarea**: ştim cu certitudine că P != EXP.** Însă nu avem nici o idee despre relaţia de egalitate între NP şi P sau între NP şi EXP**. Nu există nici o demonstraţie care să infirme că problemele din NP au algoritmi eficienţi, determinist polinomiali!** Problema P=NP este cea mai importantă problemă din teoria calculatoarelor, pentru că de soluţionarea ei se leagă o grămadă de consecinţe importante.

Probleme NP

- Nu sunt soluții polinomiale ce putem face?
  - Backtracking
    - se opreste cu solutie optima dar nu stim cand
  - Backtracking omorat (ne poate da o solutie optima)
  - Brute force (eventual omorat)
  - Rezlvare optima pentru unele cazuri particuloare (graf bipartit pentru colorare)
  -

Probleme NP

- Nu avem solutii polinomiale ce putem face?
  - Rezolvam doar pe cazuri mici
  - Rulam o perioada ne oprim
  - Găsim algoritmi care ne ofera solutii macar aproape de ce cautam

Probleme NP

- Nu sunt soluții polinomiale ce putem face?
  - Aproximare: În loc sa căutăm solutia exacta sa căutăm o solutie apropiata de solutia optima (de cel puțin 2 ori mai mare sau de cel puțin 0.001 ori mai mare).
  - Randomizare: Folosirea randomizarii pentru a obține algoritmi care rulează pe caz mediu mai rapid .
  - Restrictionare: Restrictionand datele de intrare putem obține uneori algoritmi mai eficienti (colorare pe grafuri planare/bipartite, ciclu hamiltonian pe grafuri care îndeplinesc Dirac),
  - Euristici: Algoritmi care merg bine pe multe cazuri dar pentru care nu avem soluții ca merg bine și eficient tot timpul.

Algoritmi Probabiliști

Problema rucsacului

Varianta:

Se dau o mulţime (mare) de rucsaci de capacitate egală (cunoscută, un număr natural). Se mai dă o mulţime finită de obiecte, fiecare de un volum cunoscut (număr natural). Întrebarea este: care este numărul minim de rucsaci necesari pentru a împacheta toate obiectele?

Idei ?

Idei aproximativ corecte ?

Problema rucsacului

Varianta:

Se dau o mulţime (mare) de rucsaci de capacitate egală (cunoscută, un număr natural). Se mai dă o mulţime finită de obiecte, fiecare de un volum cunoscut (număr natural). Întrebarea este: care este numărul minim de rucsaci necesari pentru a împacheta toate obiectele?

Idei aproximativ corecte ?

- **Umplem fiecare rucscac greedy**
  - **Punem la fiecare pas cel mai mare obiect pe care il putem pune…**
  - **Ne oprim cand nu putem sa mai punem nici un obiect în rucscac**
  - **Trecem la urmatorul rucsac**
- **Gradul de aproximare ?**

Problema rucsacului

Varianta:

Se dau o mulţime (mare) de rucsaci de capacitate egală (cunoscută, un număr natural). Se mai dă o mulţime finită de obiecte, fiecare de un volum cunoscut (număr natural). Întrebarea este: care este numărul minim de rucsaci necesari pentru a împacheta toate obiectele?

Idei aproximativ corecte ?

- **Umplem fiecare rucscac greedy**
  - **Punem la fiecare pas cel mai mare obiect pe care il putem pune…**
  - **Ne oprim cand nu putem sa mai punem nici un obiect în rucscac**
  - **Trecem la urmatorul rucsac**
- **Gradul de aproximare ?**
  - **2**
  - **De ce ?**

Problema rucsacului

Varianta:

Se dau o mulţime (mare) de rucsaci de capacitate egală (cunoscută, un număr natural). Se mai dă o mulţime finită de obiecte, fiecare de un volum cunoscut (număr natural). Întrebarea este: care este numărul minim de rucsaci necesari pentru a împacheta toate obiectele?

Idei aproximativ corecte ?

- **Umplem fiecare rucscac greedy**
  - **Punem la fiecare pas cel mai mare obiect pe care-l putem pune…**
  - **Ne oprim cand nu putem sa mai punem nici un obiect în rucsac**
  - **Trecem la următorul rucsac**
- **Gradul de aproximare ?**
  - **2**
  - **De ce ?**
    - **Orice rucsac (exceptand 1) va fi cel putin n/2 plin… (altfel i-am putea combina).**

Problema rucsacului

Varianta:

Se dau o mulţime (mare) de rucsaci de capacitate egală (cunoscută, un număr natural). Se mai dă o mulţime finită de obiecte, fiecare de un volum cunoscut (număr natural). Întrebarea este: care este numărul minim de rucsaci necesari pentru a împacheta toate obiectele?

Concluzie:

- **Avem solutie rapida care este în cel mai rau caz de 2 ori mai rea ca solutia optima…**
- **În practica solutia paote fi și mai buna….**
- **Putem sa incercam sa găsim solutii mai bune (cu algoritmi genetici/aproximativi) cu limitarea la aceasta solutie….**

Aproximare absoluta/relativa

Aproximare absoluta (pentru soluție de maxim) de grad C:

- Sol(i) * C >= Opt(i)

Aproximare relativa (pentru soluție de maxim) de grad C:

- Sol(i) + R >= Opt(i)

Acoperirea unui Graf

- **Care este numărul minim de varfuri care trebuie ``acoperite'' astfel ca toate muchiile dintr-un graf să fie atinse?**
  - **Idei ?**

Acoperirea unui Graf

- **Care este numărul minim de varfuri care trebuie ``acoperite'' astfel ca toate muchiile dintr-un graf să fie atinse?**
  - **Idei ?**
  - **Cuplaj maxim** 🤟💌
    - Facem cuplaj maxim, și apoi selectam toate nodurile care sunt la **ambele** capete ale muchiilor din cuplaj
    - De ce funcționează ??

Acoperirea unui Graf

- **Care este numărul minim de varfuri care trebuie ``acoperite'' astfel ca toate muchiile dintr-un graf să fie atinse?**
  - **Idei ?**
  - **Cuplaj maxim** 🤟💌
    - Facem cuplaj maxim, și apoi selectam toate nodurile care sunt la **ambele** capete ale muchiilor din cuplaj
    - De ce funcționează ??
      - Dacă am avea o muchie care nu e atinsă în nici un capăt, atunci ea putea fi adaugata la cuplaj

Acoperirea unui Graf

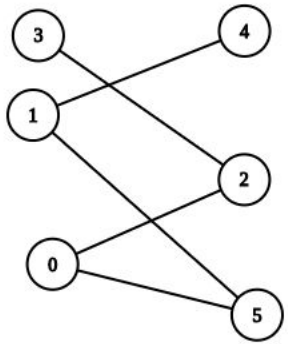
- **Care este numărul minim de varfuri care trebuie ``acoperite'' astfel ca toate muchiile dintr-un graf să fie atinse?**
  - **Idei ?**
  - **Cuplaj maxim** 🤟💌
    - Facem cuplaj maxim, și apoi selectam toate nodurile care sunt la **ambele** capete ale muchiilor din cuplaj
    - De ce funcționează ??
      - Dacă am avea o muchie care nu e atinsă în nici un capăt, atunci ea putea fi adaugata la cuplaj
    - Aproximare absoluta sau relativa ?

Acoperirea unui Graf

- **Care este numărul minim de varfuri care trebuie ``acoperite'' astfel ca toate muchiile dintr-un graf să fie atinse?**
  - **Idei ?**
  - **Cuplaj maxim** 🤟💌
    - Facem cuplaj maxim, și apoi selectam toate nodurile care sunt la **ambele** capete ale muchiilor din cuplaj
    - De ce funcționează ??
      - Dacă am avea o muchie care nu e atinsă în nici un capăt, atunci ea putea fi adaugata la cuplaj
    - Aproximare absoluta sau relativa ?
      - Relativa … de ce grad ?

Acoperirea unui Graf

- **Care este numărul minim de varfuri care trebuie ``acoperite'' astfel ca toate muchiile dintr-un graf să fie atinse?**
  - **Idei ?**
  - **Cuplaj maxim** 🤟💌
    - Aproximare absoluta sau relativa ?
      - Relativa … de ce grad ?
        - 2



Acoperirea unui Graf

- **Care este numărul minim de varfuri care trebuie ``acoperite'' astfel ca toate muchiile dintr-un graf să fie atinse?**
  - **Idei ?**
  - **Cuplaj maxim** 🤟💌
    - Facem cuplaj maxim, și apoi selectam toate nodurile care sunt la **ambele** capete ale muchiilor din cuplaj
    - De ce funcționează ??
      - Dacă am avea o muchie care nu e atinsă în nici un capăt, atunci ea putea fi adaugata la cuplaj
    - Aproximare absoluta sau relativa ?
      - Relativa … de ce grad ?
        - 2

Acoperirea unui Graf

- **Care este numărul minim de varfuri care trebuie ``acoperite'' astfel ca toate muchiile dintr-un graf să fie atinse?**
  - **Nu avem solutie aproximativa absoluta…. Sa zicem ca vrem o solutie la distanta maxim n**
    - **Pe scurt dacă exista graful G pentru care solutia noastra da solutia + 1…**
    - **Facem n+1 componente conexe identice -> solutia noastra va fi la distanta n + 1**

Algoritmi Probabiliști

- În timpul rezolvării unei probleme, putem ajunge la un moment dat în situaţia de a avea **de ales** între mai multe variante de continuare.

Algoritmi Probabiliști

- În timpul rezolvării unei probleme, putem ajunge la un moment dat în situaţia de a avea **de ales** între mai multe variante de continuare.
  - **se alege aleator una dintre variante**
  - **la executări diferite ale unui algoritm probabilist, rezultatele sunt în general diferite.**

Algoritmi Probabilişti

Categorii

- **Monte Carlo**
- **Las Vegas**
- **….**

Algoritmi Monte Carlo

Algoritmi Monte Carlo

- Furnizează totdeauna un rezultat, care însă **nu este neapărat corect**
- **Probabilitatea ca rezultatul să fie corect creşte pe măsură ce timpul disponibil creşte**

Algoritmi Monte Carlo

Se consideră un vector cu *n* elemente distincte. Să se determine un element al vectorului care să fie mai mare sau egal cu mediana a celor n numere din vector

- **n este foarte mare**
  - **timpul avut la dispoziţie este mic**

Algoritmi Monte Carlo - Mediana

max = -∞

Repetă fără a depăşi timpul disponibil:

- alegem aleatoriu x un element al vectorului
- max = maxim(max, x) = cel mai mare element ales

scrie max

Algoritmi Monte Carlo - Mediana

- Care este probabilitatea ca un element ales x să fie mai mic decât mediana?
  - 1/2

Algoritmi Monte Carlo - Mediana

- **Care este probabilitatea ca răspunsul să fie corect după k încercări?**

Algoritmi Monte Carlo - Mediana

- **Care este probabilitatea ca răspunsul să fie greșit după k încercări?**

  - Care este probabilitatea ca un element ales x să fie mai mic decât mediana?

  - Care este probabilitatea ca **toate** cele k elemente alese (în timpul de rulare avut la dispoziţie) să fie mai mici decât mediana (deci v < mediana)?

Algoritmi Monte Carlo - Mediana

- Care este probabilitatea ca răspunsul să fie corect după k încercări?

  $1 - 1/2^k$

  - **Pentru k=20, această probabilitate este mai mare decât 0,999999**
  - **Pentru k=20, această probabilitate este mai mare decât 0,999**

- **$1- (1-p)^k$**

Algoritmi Monte Carlo

Se consideră un vector cu *n* elemente. Să se determine dacă există un element majoritar în vector (cu frecvența > n/2)

Algoritmi Monte Carlo – Element majoritar

v = -∞

Repetă fără a depăşi timpul disponibil:

- alegem aleator x un element al vectorului
- Calculam f = frecventa lui x
- Daca f>n/2 scrie DA; STOP

scrie NU

Algoritmi Monte Carlo – Element majoritar

Analiza

- **Problemă de decizie**
- Dacă scrie DA, raspunsul este corect
- Dacă scrie NU, este posibil ca să existe element majoritar (deci răspunsul să fie greșit)

Algoritmi Monte Carlo – Element majoritar

Analiza

- **Problemă de decizie**
- Dacă scrie DA, raspunsul este corect
- Dacă scrie NU, este posibil ca să existe element majoritar (deci răspunsul să fie greșit)
- Care este probabilitatea de a răspunde greșit NU după k pași?

Polinom nul

Fie un polinom de mai multe variabile, x_1, x_2, ... x_n. Acest polinom poate fi descris printr-o formulă aritmetică, de pildă: $(x_1 + 1) (x_2 + 1) ... (x_n + 1)$. Întrebarea este: este acest polinom identic nul sau nu?

- Idei ?
  - Alegem seturi de valori pentru variabilele mele.
  - Dacă pentru toate seturile de valori alese avem raspunsul 0 atunci probabil e 0

Polinom nul

Fie un polinom de mai multe variabile, x_1, x_2, ... x_n. Acest polinom poate fi descris printr-o formulă aritmetică, de pildă: $(x_1 + 1) (x_2 + 1) ... (x_n + 1)$. Întrebarea este: este acest polinom identic nul sau nu?

- Idei ?
- Ce probabiltate avem dupa o testare ?
- Dar dupa mai multe ?

Algoritmi Las Vegas

Algoritmi Las Vegas

- **Nu furnizează totdeauna un rezultat**, dar dacă furnizează un rezultat atunci acesta este **corect**

- **Probabilitatea ca algoritmul să se termine creşte pe măsură ce timpul disponibil creşte**

Algoritmi Las Vegas

Se dau n texte (n foarte mare) cu următoarele proprietăţi:

- **există un unic text t_0 care apare de cel puţin 10% ori;**
- **celelalte texte sunt distincte.**

Se cere determinarea textului t_0.

Algoritmi Las Vegas - Text

Algoritm probabilist

Idee

- Generăm aleatoriu doi indici i și j și testăm dacă

$i \neq j$ și $t_i = t_j$

până când testul se încheie cu succes

Algoritmi Las Vegas - Text

```
repeat
    if LVText()
      stop
until false


LVText()
   i ← random(1..n); j ← random(1..n);
      if i≠j and t_i=t_j
           write t_i; return true
   else
```

Bibliografie

- Horia Georgescu. **Tehnici de programare**, Editura Universităţii din Bucureşti 2005

- Gilles Brassard, Paul Bratley - **Algorithmics: theory and practice**, Prentice Hall, 1988