

Var 3. 1:

Proprietate: Putem "alege" jucătorul care să aibă doar din poziții de indice par sau impar!

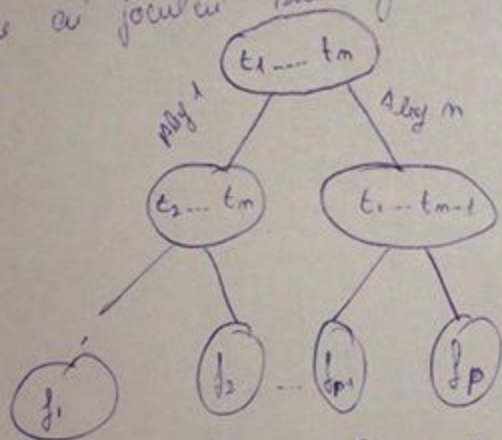
Strategie: 1) Calculăm 2 sume:  
- suma elementelor de  $n$  par  
- suma elem de  $n$  impar

2) Le comparăm

3) Deoarece noi însuși jucăm vom alege să jucăm  $n$  incluziv care compun suma maximă.

Demonstrăm proprietatea de mai sus:

Fie  $T = \{t_1, \dots, t_n\}$  tabula de joc. Codificăm poziția de desfășurare a jocului sub forma unei arbori binari în felul următor:



Considerăm cazul în care  $n$  e impar !!!

$n$  par  $\rightarrow$  Analog.

Nu facem prima alegere.  $(1+n)$  este un număr par. Dacă alegem  $n$ , alegem jucătorul 2 să aibă dintre 1 și  $n-1$  adică număr impar.

Dacă alegem 1, el alege să aibă dintre 2,  $n$  adică 2 nr par.

Deoarece strategia e ca noi să ne putem purta par sau impar pe tot parcursul jocului alegând jucătorul care alege poziția desfășurării.

Alegerea greedy este să alegem să includem ce compun suma maximă.

Complexity:  $O(n^2)$  because a particle a given center  
interacts with all other particles in solution.  
n = number of particles

⇒ Complexity is  $O(n^2)$ .

Variantă 3, pb 2:

pseudocod:

$S = \emptyset$

// Soluție

$G = (V, E)$

$F = \{v \in V \mid v \text{ frunză}\}$

cot timp  $F \neq \emptyset$

$S = S \cup F$

~~F = F \setminus \{v\}~~

Elimină părinți ( $F$ );

$F = \{p \mid p \text{ părinte}$

Actualizare  $\text{Grade}(v)$ ;  
pt  $v \in F, v \in F, p \in V$

Return  $S$ ;

Soluție: Selecționăm la fiecare pas frunzele arborelui și le adăugăm la soluție, și eliminăm părinții frunzelor din mulțimea de noduri ne selectate. Se face acest lucru până când rămânem fără noduri vizitate;

~~Am~~ Observații

Algoritm greedy constă în faptul că alegem să adăugăm la soluție doar frunze cu un singur nod.

pentru că frunzele sunt adiacente

Corectitudinea:

Algoritmul generează o soluție întotdeauna deoarece avem de ales între frunze întotdeauna un număr finit de frunze, și că le putem adăuga. Algoritmul se termină când nu mai sunt frunze nemarkate.

Demonstrăm că alegând doar frunze ajungem la soluție optimă:

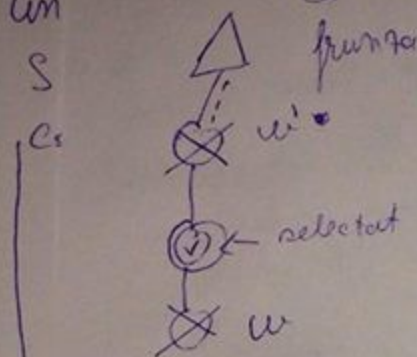
Fie  $S$  mulțimea de noduri adăugate până la pasul curent

Fie  $v$  nodul selectat la acest pas.



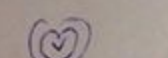
Varia  
p6' cazul 1:  $v \in \text{pronto} \rightarrow$  se poate un nr minim de module (adica doar unul) - cind se alege  $v$ .

Per cazul 2:  $v$  e adiacent cu maxim 2 prieteni  $\Rightarrow$  L. fel  
cu  $v$  ca in cazul 1 se pierde un minim de noduri.



Chiar daca se pierd 2 noduri la ologra  
la V este tot o pierdere minima pt  
ca daca nu s-ar fi pierdut un n  
pierdea ~~de~~ datul lui. ~~off~~

Cazul 3:  $V$  e adiacent cu minimum 2 frunze. În acest caz dacă alegem  $V$  pierdem un număr mare de noduri și nu reușim să ajungem la soluție optimă.



Deoarece alg. abele erau foarte rezul to co ~~tr~~  
 la fiecare pas se pierde un nr minim de noduri.  
~~reduceri similare~~ <sup>NU</sup> poate fi aplicat

Un algoritm de tip greedy, similar <sup>NU</sup> poate fi aplicat  
si pe un graf neorientat deoarece ~~nu este un caz~~  
~~si nu este un caz al grafului / Dar uneori nu se poate!!~~

4) Se face o 2 colorări a graficului / Dar uneori nu se poate!!

2) Se alegă cuburile care au nr maxim de moduri  
modurile din cuburile alese la soluție.

2) Se alega cultura  
3) Se alega modurile din cultura aleasă la soluție.

Complexitate alg. inițial:  $O(n)$  deoarece parcurgem nodurile arborelui a singuri de o dată.

Variantă 3

P62. Continuare

Pentru un graf neorientat care nu poate conține  
un algoritm greedy în felul următor:

$S = \emptyset$

Cot timp mai sunt noduri nevizitate executo:

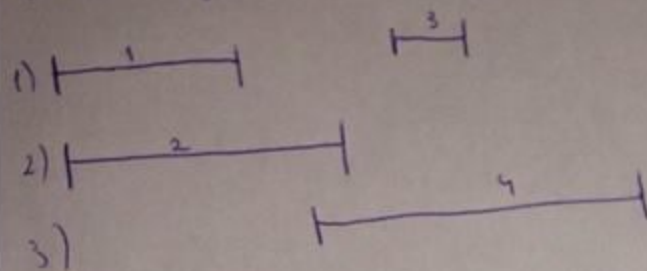
Algoritm la  $S$  nodurile care au gradul minim

- Eliminați nodurile adiacente imediate cu prietenii la  
pentru ei nu mai fi luate în considerare la următorul  
pas

- Actualizează grade.

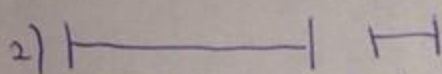
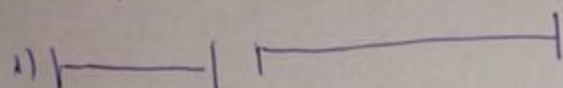


3.  
b) Dacă algoritmul ordonează intervalele în ordine urmatoare :



$\Rightarrow$  Nu este corect !

Soluție :



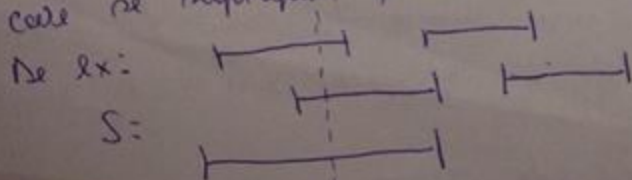
a) Algoritmul este corect.

Dem: Fie  $d$  numărul de clase alocate de algoritmul  
Clasa  $d$  a fost alocată pt.  $t_0$  la un pas a fost  
memorie  $t_0$  se programase o activitate  $j$  care începe la  
 $t_j$  și se termină la  $t_j$ . Activitatea  $j$  a fost incompatibilă  
cu toate celelalte  $d-1$  clase ~~pt.  $t_0$~~  erau disponibile  
~~după timpul~~ mai târziu de  $t_j$ .  
Deoarece am sosit după timpul de început, aceste  
incompatibilități sunt cauzate de activități care încep  
mai devreme de  $t_j \Rightarrow$  Sunt  $d$  activități  
care se suprapun la timpul  $t_j + \epsilon$

Mai formal:

Fie  $S$  mulțimea de intervale date ca input.

Numim adămușul lui  $S$  numărul maxim de intervale  
care se suprapun peste același punct (oro).



În exemplul de mai sus  $S$  are adămușul 3.

~~Problema~~ Avem fiecare multitudine din  $S$  o etichetă. Astfel pt. a reduce la a determina nr. minim de etichete necesare pt. o.s. oricare  $t$  ar fi două intervale cu aceeași etichetă, ele sunt disjuncte (compatibile).

Afirmatie: Soluția construită de algoritmul este posibilă.

Dem: Algoritmul ia pe rând fiecare interval  $I_i$  și asoc.

o etichetă (il adaugă la o submulțime), iar atunci când nu îl poate atribui o etichetă deja existentă se crează una nouă. (1)

Propoziție: Nr. max de clase necesare este  $\geq$  cu adâncimea lui  $S$ .

Dem: Fie  $d$  adâncimea lui  $S$ .

Dim model în care am definit adâncimea  $\Rightarrow$

$\exists I_1, \dots, I_d$  intervale care se suprapun (au etichete diferite). Deoarece toate cele  $d$  intervale se suprapun trebuie să folosim câte o etichetă dif. pt. fiecare.

Deci alg. folosește cel puțin mai multe etichete. (2)

~~Algoritmul~~ Se dem. în continuare că alg. folosește exact  $d$  etichete și folosind punctul (2) va rezulta optimalitatea.

Fie  $t$  nr. de etichete fol. la pasul curent (inițial  $t=0$ ), arătăm că la fiecare pas  $t \leq d$ .

Fie  $I_j$  intervalul selectat de alg. la pasul curent.

Cazul 1: Lui  $I_j$  i se poate adăuga o etichetă existentă  $\Rightarrow t$  nu se modifică.

Cazul 2: Lui  $I_j$  nu i se poate adăuga o etichetă  $\Rightarrow I_j$  se suprapune cu  $t$  intervale într-un pt.  $\Rightarrow$  sunt  $t+1$  intervale (incluzând  $I_j$ )

care se suprapun într-un pt., din faptul că toate cele  $t$  intervale

au extremități inițiale mai mici decât  $I_j$ , sau egale decât

cea a lui  $I_j$ . Punctul în care se suprapun este chiar extremitatea inițială a lui  $I_j$ .  $\Rightarrow t+1 \leq d$  (3)

Dim (1) (2) (3)  $\Rightarrow$  Alg. este optim



2) Ca la punctul a) se asociază fiecărui interval o etichetă.  
 Demonstrăm că nr. de etichete necesare este egal cu adâncimea lui  $S$ .  
 Considerăm cazul când alg nu poate atribui o etichetă dintr-un  
 anumit interval selectat la pasul curent, fie el  $I_j$ .  
 Fie  $t$  nr. de etichete folosite până la acest pas.  $\Rightarrow I_j$  este interval  
 care se suprapune dintr-o parte dintr-un

Se consideră din fiecare submulțime intervalul cu  
 cea mai ~~mare~~ extremitate finală. Fie  $m$  cea mai mică

extremitate finală a unui astfel de interval. Demonstrăm  
 că există în fiecare ~~submulțime~~ submulțime un interval care  
 conține  $m$ .

P.p. alegem ca  $I$  un interval o submulțime  $A \subseteq S$  în  
 care nu există nici un interval care conține  $m$ . fie  $I_k$   
 intervalul cu extremitate finală cea mai apropiată de  $m$

Cașul 1:  $I_k$  este ultimul interval adăugat în

$A \Rightarrow I_k \not\subset m$   $\&$  am presupus că  $m$  e minimul.

Cașul 2:  $\exists I_{k+1}$  a.f.  $D_{k+1}$  (extremitatea lui  
 inițială) e mai mare ca  $m$ .  $\&$  Algoritmul  
 ar fi ales să îl pună pe  $I_{k+1}$  în mulțime  
 care conține intervalul care se termină în  $m$ , deoarece  
 $m > I_k$  iar algoritmul alege tot submulțime  
 care conține intervalul cu ext. finală cea mai  
 mare.



d) S-a demonstrat la pas ca alg. de selectie a unui cardinal maxim de activitati e corect.

Dupa ce ruleaza o algoritmului se creeaza o submultime  $A \subseteq S$  de cardinal maxim. ~~Care este ml, o alt interval~~  
~~Care nu este in S sau~~

Daca un interval  $I$  nu este in  $A \Rightarrow \exists J$  un alt interval a.i.  $I$  nu e compatibil cu  $J$ .  $\Rightarrow$   
Algoritmul genereaza intotdeauna o solutie, iar solutia generata este optima deoarece fiecare subml,  $A_i$  generata,

$A_i \subseteq S$  este de cardinal maxim.