Tehnica Greedy

Enunt general:

Să considerăm o mulţime A cu n elemente. Se cere o submulţime S a sa cu m elemente, astfel încât să fie îndeplinite anumite condiţii.

- în cazul m=n este importantă ordinea alegerii elementelor
- condiţiile diferă de la o problemă la alta.

Exemplu. Problema Suma maximă.

1. Se consideră o mulţime A cu n numere reale. Se cere o submulţime S a sa, astfel încât suma elementelor sale să fie maximă.

Exemplu. Pentru numerele 1 -5 6 2 -2 4 răspunsul este 1 6 2 4 (suma 13).

Rezolvare.

1. Initial submultimea solutie căutată este vidă S= Φ
2. Se alege un prim element x al mulţimii de numere reale. Dacă este posibil (adică dacă x>0), acesta va fi adăugat soluţiei S=S U {x}
3. Se alege un al doilea număr, cu care se procedează în mod asemănător.
4. Algoritmul se încheie când au fost alese şl eventual adăugate toate elementele mulţimii.

Pentru a rezolva o problemă cu Greedy, soluţia se construieşte, după regula:

```
S ← Ø
┌cât timp (nu s-a obtinut solutia S) si (mai există elemente neselectate in A)
|      Alege x din A  // “alege cel mai bun element x,  disponibil in A”
|      ┌dacă   S U {x} satisface restrictiile problemei
|      |       atunci   S ← S U {x}     // “adauga x la S”
|      └■
└■
```

Obs.

- Alegere lui x din A se face după un mecanism specific fiecărei probleme în parte
- Algoritmul se termină fie când a fost găsită soluţia cerută, fie când s-a constatat inexistenţa acesteia.
- Tehnica nu poate fi standardizată (nu există o schemă unică de rezolvare a problemelor)
- Există putine probleme care pot fi rezolvate cu această tehnică
- Solutia este construită pas cu pas ca la *Backtracking, fără mecanismul de revenire la pasul anterior.*
    - ambele tehnici oferă soluţii sub formă de vector;
    - tehnica Backtracking poate oferi toate soluţiile problemei, în timp ce tehnica Greedy oferă o singură soluţie => tehnica Greedy nu dispune de mecanismul întoarcerii, specific tehnicii Backtracking ($O(n^m)$. :
- *Pentru fiecare problemă în parte, după ce se identifica un algoritm, este obligatoriu să se demonstreze că acesta conduce la soluţia optima. Demonstraţia faptului ca se ajunge la soluţia optimă este specifică fiecărei probleme în parte.*
- *Tehnica Greedy conduce la timp de calcul polinomial.*

- Să presupunem că mulţimea din care se face alegerea are n elemente si că soluţia are tot n elemente (caz maxim). Se fac n alegeri, la fiecare alegere se fac n teste, rezulta un algoritm cu timp polinomial $O(n^2)$. (Back-timp exponential)
- De multe ori, este necesar ca elementele mulţimii A să fie mai întâi sortate si apoi să alegem din acestea. Sortarea necesita un timp minim $O(n \times \log_2 n)$. Cum sortarea se efectuează la început timpul de calcul este $n \log_2 n + n^2$. Prin urmare, acest timp se adună, deci nu influenţează rezultatul(= $O(\max(n^2, n \times \log_2 n))$
- Ordinul de complexitate al algoritmilor de tip "greedy" este (în functie de natura elementelor din A si algoritmul de sortare utilizat): $O(n^2)$ sau O(nlgn) sau O(n) ➔ Algoritmii de tip greedy sunt eficienti

Întrebare: *fiind dată o problemă, există întotdeauna un algoritm de tip Greedy care găseşte soluţia optimă?*
Răspuns: Evident, NU. Există probleme pentru care nu se cunosc astfel de algoritmi. Mai mult, pentru cele mai multe probleme nu se cunosc algoritmi Greedy.

Obs.
- Avantajul timpului polinomial, conduce la necesitatea utilizării tehnicii Greedy.
- Nu tuturor problemelor li se pot aplica algoritmi de acest tip.
- *Pentru problemele pentru care nu se cunosc algoritmi care necesită timp polinomial, se caută soluţii, chiar dacă nu optime, dar apropiate de acestea si care au fost obţinute în timp util.*
  Multe din aceste soluţii sunt obţinute cu Greedy.
- Astfel de algoritmi se numesc ***algoritmi euristici.***

Verificarea corectitudinii

Multe dintre problemele pentru care solutia Greedy este optimă sunt caracterizate prin următoarele proprietăti:

- Proprietatea substructurii optime
  - orice solutie optimă a problemei initiale contine o solutie optimă a unui subprobleme (problema de acelasi tip dar de dimensiune mai mică)
  - Cand se poate spune despre o problema ca are proprietatea de substructura optima?
    - Atunci cand pentru o solutie optima $S=(s_1,\ldots,s_k)$ a problemei de dimensiune n, subsetul $S_{(2)}=(s_2,\ldots,s_k)$ este o solutie optima a unei subprobleme de dimensiune (n-1).
  - Cum se poate verifica daca o problema are aceasta proprietate ?
    - Demonstratie prin reducere la absurd

- Proprietatea alegerii Greedy
  - Componentele unei solutii optime au fost alese folosind criteriul Greedy de selectie sau pot fi înlocuite cu elemente alese folosind acest citeriu fără a altera proprietatea de optimalitate
  - Când se poate spune ca o problema are proprietatea de alegere "Greedy"?
    - Atunci când solutia optimă a problemei fie este construită printr-o strategie "greedy" fie poate fi transformată într-o altă solutie optimă construită pe baza strategiei "greedy"
  - Cum se poate verifica dacă o problema posedă sau nu această proprietate?
    - Se demonstrează că înlocuind primul element al unei solutii optime cu un element selectat prin tehnica "greedy", solutia rămâne optimă. Apoi se demonstrează acelasi lucru pentru celelalte componente fie folosind metoda inductiei matematice fie folosind proprietatea de substructura optimă

Probleme

1. *Problema Suma maximă (Greedy optim)*

Se consideră o mulţime A cu n numere reale. Se cere o submulţime S a sa un număr maxim de elemente, astfel încât suma elementelor sale să fie maximă.

Exemplu. Pentru numerele 1 -5 6 2 -2 4 răspunsul este 1 6 2 4 (suma 13).
Soluție: Se aleg nuerele pozitive

2. *Problema 2 suma maximă(Greedy optim)*

Se consideră o mulțime de n numere reale. Se cere o submulțime a sa cu m elemente, astfel încât suma elementelor din submulțimea S să fie maximă.

Soluție. Fie $A=(a_1>=a_2>=\ldots>=a_n)$. Solutia greedy este $(a_1,\ldots,a_m)$.

Verificare corectitudine.
Fie O=(o1,…,om) o solutie optimă.

a) Proprietatea de alegere "greedy". Presupunem că $o_1 \neq a_1$. Cum a_1 este cel mai mare număr din A atunci $o_1<a_1$. Înlocuind în O pe o_1 cu a_1 obtime o altă solutie $O'=(a_1,o_2,\ldots,o_m)$ cu proprietatea: $a_1+o_2+\ldots+o_m> o_1+o_2+\ldots+o_m$
Adica O' este mai bună decât O. Contradictie cu pp că O este optimă. Deci o_1 trebuie să fie egală cu a_1

b) Proprietatea de substructura optimă. Se demonstrează prin reducere la absurd.
Presupunem că $(o_2,\ldots,o_m)$ nu este solutie optimă pentru subproblema corespunzătoare lui $A_{(2)}=(a_2,\ldots,a_n)$.
Considerăm că $O'_{(2)}= (o'_2,\ldots,o'_m)$ este solutie optimă pentru această subproblemă.
Atunci $O'=(a_1, o'_2,\ldots,o'_m)$ este o solutie mai bună decât O. Contradictie…deci problema are proprietatea de substructură optimă

3. *Problema planificării spectacolelor(Greedy optim)*

Managerul artistic al unui festival trebuie să selecteze o multime cât mai mare de spectacole ce pot fi jucate în singura sală pe care o are la dispozitie. Stiind că i s-au propus n<=100 spectacole si pentru fiecare spectacol k din cele n cunoaste intervalul de timp în care se poate desfăsura $[s_k, f_k)$ (s_k reprezintă ora si minutul de început, iar f_k ora si minutul de terminare al spectacolului k)
Scrieti un program care să permită spectatorilor vizionarea unui număr cât mai mare de spectacole în aceeasi zi.

De exemplu, dacă vom citi n=5 si următorii timpi:

12 30 16 30
15 0 18 0
10 0 18 30
18 0 20 45
12 15 13 0

Spectacolele selectate sunt: 5 2 4.

Solutie

P1. Ordonăm spectacolele crescător după ora terminării lor.

P2. Selectăm initial primul spectacol (deci cel care se termină cel mai devreme).

P3. Selectăm primul spectacol neselectat, care nu se suprapune cu cele deja selectate (deci cel care începe după ce s-a terminat ultimul spectacol selectat)

P4. Dacă nu se mai pot face alegeri, algoritmul se termină. Altfel, se selectează spectacolul găsit si se reia algoritmul de la pasul P3.

```
sortare descrescatoare dupa terminare a spectacolelor (s[1..n])
sol[1]←a[1]
k←1
┌pentru i←2,n execută
|  ┌dacă s[i].inceput>=sol[k].terminare
|  |       atunci  k←k+1;  sol[k]←s[i]
|  └■
└■
scrie sol[1..n]
```

Verificare corectitudine.

- Pp ca setul de spectacole este ordonat crescator dupa momentul de terminare (s_1.terminare< s_2.terminate<…<s_k.terminare).

- Proprietatea de alegere "greedy": Fie O=(o_1,o_2,…,o_m) o solutie optimă si X=(x_1,x_2,…,x_k) solutia dată de algoritm
- Dacă k>m atunci O nu este soluía optimă.
- Dacă k=m atunci X este optimă
- Dacă k<m , în acest caz putem înlocui în O pe s_1 cu x_1 (spectacolul care se termină cel mai repede) fără a altera restrictia problemei (spectacolele selectate respectă cerintele) si păstrând acelasi număr (maxim) de spectacole selectate. Obtinem solutia optimă O'=(x_1,o_2,…,o_k)

- Proprietatea de substructură optimă. Considerăm solutia optimă O'= (x_1,o_2,…,o_k). Pp că (o_2,…,o_k) nu solutie optimă pt subproblema selectiei din {s_2,s_3,…,s_n}. Rezultă că există O"=(o''_2,…,$o''_{k''}$) o altă solutie cu k">k. Acest lucru ar conduce la o solutie (x_1,o''_2,…,$o''_{k''}$) mai bună decat O'=(x_1,o_2,…,o_k). Contradictie.

4, Problema rucsacului – cazul continuu *(Greedy optim)*

Se consideră un set de n obiecte si un rucsac de capacitate dată GMax. Pentru fiecare obiect k se cunoaste greutatea gk si câstigul ck care poate fi obtinut prin transportarea acestuia cu rucsacul până la o destinatie fixată.
Să se determine o încărcare optimală a rucsacului, în ipoteza în care orice obiect poate fi încărcat întreg în rucsac sau partial.

De exemplu, pentru n=5, GMax=100 si câstigurile-greutătile următoare:

`(1000 120)(500 20)(400 200)(1000 100)(25 1)` se va afisa pe ecran:

```
2 100.00%
4 79.00%
5 100.00%
```

Solutie

Vom reprezenta o solutie a problemei ca un vector x, cu n componente, în care retinem pentru fiecare obiect fractiunea încărcată în rucsac. Deci vectorul x trebuie să îndeplineasc următoarele conditii:

1). $xi \in [0,\ 1], \forall i \in \{1, 2, ..., n\}$.

2). $g1 \cdot x1 + g2 \cdot x2 + \ldots + gn \cdot xn \le Gmax$

3). $f(x) = c1 \cdot x1 + c2 \cdot x2 + \ldots + cn \cdot xn$ este maximă.

Algoritmul:

P1. Ordonăm obiectele descrescător după câstigul pe unitatea de greutate (valoare care constituie o măsură a eficientei transportării obiectelor).

P2. Cât timp este posibil (încap în rucsac), selectăm în ordinea dată obiectele în întregime.

P3. Completăm rucsacul cu un fragment din următorul obiect ce nu a fost selectat

```
sortare descrescatoare a obiectelor după câstigul pe unitatea de greutate c[i]/g[i]
k←1
┌cât timp GMax>0 si i<=n execută
|   ┌dacă Gmax>=g[i]
|   |       atunci  sol[k]←100;  Gmax←Gmax-g[i]
|   |       altfel   sol[k]←100*Gmax/g[i]; Gmax←0
|   └■
|    i←i+1
└■sol[1]←a[1]
scrie sol
```

5. Problema 3 de maxim (Greedy optim)

Se consideră multimea A cu m elemente numere întregi nenule {a1,a2,..,am} si multimea B cu n elemente numere întregi nenule, n>=m. Să se determine un sir cu melemente din B : x1,x2,...,xm de elemente astfel încât suma următoare să fie maximă S=a1*x1+a2*x2+...+am*xm.

Solutie.

1. Se sortează crescător elementele lui A, respectiv B={b1,b2,...,bn}.
2. Dacă n=m atunci x[i]=b[i] , i=1,...,m
3. Dacă n>m atunci

3.1. daca toate 0<a[1] atunci x[i]=b[n-m+1], x[2]=b[n-m+2],…,x[m]=b[n]

3.2. daca a[1]<=a[2]<=…<=a[n]<0 atunci x[1]=b[1], x[2]=b[2],…,x[m]=b[m] (primele m elemente din B)

3.2. daca A are p elemente negative si t elemente pozitive (p+t=m) a[1]<=a[2]<=…<=a[p]<0<a[p+1]<…<a[m] atunci x[1]=b[1], x[2]=b[2],...,x[p]=b[p] si x[p+1]=b[n-t+1],..., x[m]=b[n] (primele p elemente din B si ultimile t)

6. *Problema banilor*
(Greedy euristic - soluție obținută nu este întotdeauna optimă)

Scrieţi un program, care afişează modalitatea de plată, folosind un număr minim de bancnote, a unei sume întregi ***S*** de lei (S<20000). Plata se efectuează folosind bancnote de n tipuri distincte cu valorile b1=1 leu, b2,...bn, cu valoarea de lei. Din fiecare tip de bancnote avem la dispoziție un număr nelimitat.

Intrare: Fişierul text ***BANI.IN***
67 (=S)
3 (=n)
1 5 10

Ieşire: 6x10+1x5+2x1

Soluție. Se sortează bancnotele după valoare descrescător. Suma este plătită, cât e posibil, cu bancnote de primul tip, apoi cu al doilea tip,...,până la achitarea în totalitate.

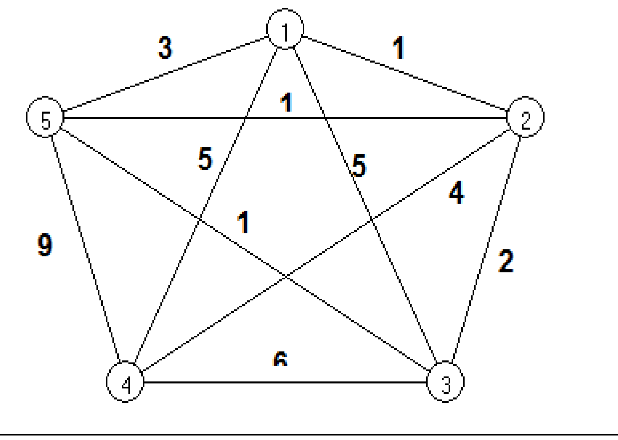
Soluția nu este intotdeauna optimă. Ex. Pentru S=10 si tipurile 1,3,4, algoritmul dă soluția: 2x4+2x1 adică 4 bancnote. O soluție cu 3 bancnote: 1x4+2x3.

7.Săritură calului pe tabla de șah (Greedy euristic)

Se dă o tablă de șah cu dimensiunea nxn. Un cal se află în poziția (1,1). Scrieţi un program care să afișeze un șir de mutări ale calului astfel încât acesta să acopere întreaga tablă fără a trece printr-o căsuță de două ori.
Soluție. La fiecare pas se alege acea mutare (dintre cele 8 posibile) care așează calul în poziția din care vom avea cel mai mic număr de variante de mutare.

8.Problema comisului voiajor (Greedy euristic)

Un comis voiajor pleacă dintr-un anumit oraș și trebuie să viziteze un număr de orașe și să se întoarcă în orașul din care a plecat, cu un efort minim.
Orice oraș i este legat de orice alt oraș j printr-un drum de A[i][j] km. Se cere traseul pe care trebuie sa-l parcurg comisul astfel încât să parcurgă un număr minim de kilometri.



matricea A

```
0 1 5 5 3
1 0 2 4 1
5 2 0 6 1
5 4 6 0 9
3 1 1 9 0

O soluție: 1,2,5,3,4,1
lungime drum=14
```

OBS. Dacă nu toate orașele sunt legate între ele printr-o șosea putem considera că dintanța dintre ele este un număr foarte mare (∞ km).

Soluție. Pas 1. Se alege un oraș de pornire.
Pas 2...,n-1. Se selectează orașul cel mai apropiat de orașul ales anterior.

9. Colorarea hărților **(Greedy euristic pt număr minim de culori)**

Sunt date N țări precizându-se relațiile de vecinătate. Se cere o posibilitate de colorare a hărții celor N țări astfel încât să nu existe țări vecine colorate la fel.

Soluție.
Codificăm culorile cu numerele 1,2,3,....
Colorăm țara 1 cu culoarea 1.
Presupunem colorate primele K-1 țări. Țara k va fi colorată cu culoarea cu cel mai mic număr astfel încât nici una din țările vecine să nu fie colorate la fel.
Soluția nu e optimă întotdeauna.

10. Depozit (Greedy)

Considerăm un depozit cu n (n≤1000) camere, care contin respectiv cantitătile de marfă C1, C2, …, Cn, (numere naturale). Scrieti un program care să determine un grup de camere cu proprietatea că suma cantitătilor de marfă pe care le contin se poate împărti exact la cele n camioane care o transportă.

Solutie
Problema este particulară: solicită determinarea unei submultimi a unei multimi cu n elemente, divizibilă tot cu n. Vom construi sumele S1, S2, ..., Sn si resturile pe care acestea le dau prin împărire la n astfel:

```
S1=C1 → R1=S1 % n
S2=C1+C2 → R2=S2 % n
...
Si=C1+C2+...+Ci → Ri=Si % n
...
Sn=C1+C2+...+Cn →  Rn=Sn % n
```

Cazul I: Există un rest `Ri=0`. În acest caz suma `Si` este divizibilă cu `n`, prin urmare camerele solicitate sunt `1`, `2`, ..., `i`.

Cazul II: Toate resturile sunt diferite de `0`. Prin urmare `R1`, `R2`, ..., `Rn` sunt `n` resturi care iau valori în multimea {`1`, `2`, ..., `n-1`}. În mod obligatoriu există cel puțin două resturi egale: Ri=Rj (`i<j`), adică `Si` si `Sj` produc acelasi rest la împărtirea cu `n` → suma `Sj-Si` este divizibilă cu `n`, deci camerele solicitate sunt `i+1`, `i+2`, ..., `j`.
Observatii
1. Solutia nu este unică. Procedeul prezentat produce o solutie posibilă.
2. Rezolvarea se bazează pe *Principiul cutiei al lui Dirichlet*

11. Problema instructorului de schi (Greedy)

Un instructor de schi are la dispoziie n perechi de schiuri (n<50) pe care trebuie să le distribuie la n elevi începători. Scriei un program care să distribuie cele n perechi de schiuri astfel încât suma diferenţelor absolute dintre înălțimea elevului si lungimea schiurilor atribuite să fie minimă.
Solutie.
Se sorteaza crescător schiorii după înalțime. Se sortează crescător schiurile după lungime.
Asociem primul schior cu prima pereche de schiuri din sirurile sortate, apoi al doilea schior cu a doua pereche de schiuri... etc

12. Zig-zag (Greedy)

Din fisierul `ZigZag.in` se citesc `N` numere întregi (`N≤1000`). Afisați pe prima linie a fisierului `ZigZag.out` cel mai lung MZigZag care se poate construi din numerele citite. Numim MZigZag o secvență de numere $a_1, a_2, ..., a_m$ astfel încât $a_1 \leq a_2 \geq a_3 \leq a_4 \geq a_5 \leq ... a_{m-1} \geq a_m$.
De exemplu:

ZigZag.in	**ZigZag.out**
7	
7 5 0 1 4 9 3	0 9 3 7 1 5 4

Soluție.
Sortăm crescător șirul. Dacă n=par, eliminăm ultimul număr din șir si n←n-1.
Fie s[1]<=s[2]<=...<=s[n] șirul sortat.
Construim un nou șir astfel: s[1], s[n], s[2], s[n-1],...s[n/2], s[(n+1)-n/2], s[n/2+1]

13. Interclasarea optimală a n șiruri (Greedy optim)

Se consideră n șiruri de numere ordonate crescător S1,S2,...,Sn, având lungimile L1, L2, ...,Ln. Să se interclaseze șirurile date și să se obțină un nou șir cu lungimea L1+L2+L3+...+Ln, astfel încât numărul de comparații între elementele șirurilor date să fie minim.

Soluție.Presupunem ca avem 3 șirule S1,S2 si S3 cu lungimile 40, 50,30.

- Pentru interclasarea lui S1 cu S2 numărul de comparații este 40+50=90. Daca interclasam șirul obținut cu S3 se vor mai face încă 90+30=120 interclasări. In total 90+120=210 comparatii
- Pentru interclasarea lui S1 cu S3 numărul de comparații este 40+30=70. Daca interclasam șirul obținut cu S2 se vor mai face încă 70+50=120 interclasări. In total 70+120=170 comparatii
- Pentru interclasarea lui S2 cu S3 numărul de comparații este 50+30=80. Daca interclasam șirul obținut cu S1 se vor mai face încă 80+40=120 interclasări. In total 80+120=200 comparatii
- Pentru exemplul dat, interclasarea în ordinea S1,S3,S2 se face cu minimum de comparații 190.

Concluzie.
Inițial sunt n șiruri.Se interclaseaza primele 2 șiruri cele mai scurte.
Din cele n-1 șiruri se selectează primele 2 cele mai scurte care se vor interclasa.....etc
La fiecare pas se interclasasează cele mai scurte 2 șiruri până la final.

14. Reactivi (Greedy optim)

Într-un laborator de analize chimice se utilizează N reactivi. Se stie că, pentru a evita accidentele sau deprecierea reactivilor, acestia trebuie să fie stocati în conditii de mediu speciale. Mai exact, pentru fiecare reactiv x, se precizează intervalul de temperatură [minx, maxx] în care trebuie să se încadreze temperatura de stocare a acestuia. Reactivii vor fi plasati în frigidere. Orice frigider are un dispozitiv cu ajutorul căruia putem stabili temperatura (constantă) care va fi in interiorul acelui frigider (exprimată într-un număr întreg de grade Celsius). Scrieti un program care să determine numărul minim de frigidere necesare pentru stocarea reactivilor chimici.

Date de intrare
Fisierul de intrare react.in contine:
– pe prima linie numărul natural `N`, care reprezintă numărul de reactivi;
– pe fiecare dintre următoarele N linii se află min max (două numere întregi separate printr-un spatiu);
– numerele de pe linia x+1 reprezintă temperatura minimă, respectiv temperatura maximă de stocare a reactivului `x`.

Date de iesire
Fisierul de iesire `react.out` va contine o singură linie pe care este scris numărul minim de frigidere necesar.

Restrictii
- 1 ≤ N ≤ 8000
- -100 ≤ min_x ≤ max_x ≤ 100 (numere întregi, reprezentând grade Celsius), pentru orice x de la 1 la N
- un frigider poate contine un număr nelimitat de reactivi

Exemple

Exemplul 1		Exemplul 2		Exemplul 3	
reactivi.in	**reactivi.out**	**reactivi.in**	**reactivi.out**	**reactivi.in**	**reactivi.out**
3 -10 10 -2 5 20 50	2	5 -10 10 10 12 -20 10 7 10 7 8	2	4 2 5 5 7 10 20 30 40	3

Solutie.
Intervalele de temperatură pot fi considerate segmente de dreaptă. Problema se reduce ladeterminarea unui număr minim de puncte astfel încât orice segment să contină cel putin unul dintre punctele determinate.

Se sortează în primul rând intervalele de temperatură crescător după temperatura minimă si descrescător după temperatura maximă.
Se deschide un frigider si se plasează primul reactiv în acest frigider. Pentru frigiderul curent se reține temperatura minimă si temperatura maximă (intervalul de temperatură în care poate fi setat).
Se parcurg succesiv reactivii (în ordine) si pentru fiecare reactiv se verifică dacă el poate fi plasat în frigiderul curent (pentru aceasta, trebuie ca intersectia dintre intervalul de temperatură al frigiderului si intervalul de temperatură al reactivului să fie nevidă). Dacă da, se plasează acest reactiv în frigiderul curent (actualizând corespunzător intervalul de temperatură al frigiderului). În caz contrar, se deschide un nou frigider (intervalul de temperatură al acestui frigider va fi intervalul reactivului plasat în el).

15. Lungimi de interval *(Greedy optim)*

Se dau N intervale **[A_i,B_i]** (1 ≤ i ≤ N). Calculati suma lungimilor tuturor intervalelor. Intervalele care se suprapun se vor lua in considerare o singura data.

Date de Intrare

Fisierul de intrare linterv.in va contine mai multe teste. Pe prima linie se va afla T numarul de teste. Pe prima linie a fiecarui test se va afla N - numarul de intervale, urmand N linii cu cate doua numere A_i si B_i - capetele intervalelor.

Date de Iesire

Fisierul de iesire linterv.out va contine T linii pe fiecare aflandu-se un singur numar x - suma calculata.

Restrictii

- 1 ≤ N ≤ 5.000
- -1.000.000 ≤ A_i ≤ B_i ≤ 1.000.000
- 1 ≤ T ≤ 75

Exemplu

linterv.in	linterv.out
1 6 -5 5 0 3 2 8 10 13 11 15 100 100	18

Lungimea intervalului [a,b] este b-a prin urmare avem:
[-5,5] -> 5-(-5) = 10
[0,3] - este inclus in [-5,5], nu il numaram
[2,8] - [2,5] este inclus in[-5,5], nu il numaram, mai ramane 8-5 = 3
[10,13] - nu e inclus nicaieri, 13-10 = 3
[11,15] - [11,13] este inclus in [10,13], mai ramane 15-13 = 2
[100,100] - are lungimea 0
Adunand obtinem 10+3+3+2 = 18

16.Reuniune de intervale *(Greedy optim)*

Se consideră n intervale închise [a,b] , a,b numere întregi, n,a,b<100000. Să se determine reuniunea acestora. Exemplu: pentru n=5 si intervalele:

2 4 **1 3** **5 8** **10 12** **6 9**	**se va afisa:** **1 4** **5 9** **10 12**

Indicatie: se ordonează intervalele după extremitatea stângă. Printr-o parcurgere liniară a sirului intervalelor, se actualizează pas cu pas capatul stâng si cel drept al intervalului reuniune.

17.Eliminări de intervale *(Greedy optim)*

Se consideră un sir de n intervale [ai,bi], cu ai, bi numere întregi. Un interval [c,d] poate fi eliminat din sir dacă există un alt interval [x,y] din sir care să- includă: [c,d]⊂[x,y]. Determinatí numărul maxim de intervale care pot fi eliminate.

Restrictii:

- n<16000
- ai,bi<2000000000

Exemplu>

7 **0 10** **2 9** **3 8** **1 15** **6 20** **1 7** **2 5**	**se va afisa:** **4**

Indicatie: se ordonează crescător intervalele după extremitatea stângă si descrescător după extremitatea dreaptă pt extremităti stângi egale. Printr-o parcurgere liniară a sirului intervalelor, se actualizează pas cu pas capatul stâng si cel drept al intervalului reuniune.

La parcurgerea sirului dublu-sortat se va actualiza cel mai mare capăt din dreapta maxd (initial maxd=extremitatea dreapta a primului interval). Un interval cu extremitatea dreaptă
d < maxd este inclus intr-ul altul din sir, deci va fi eliminat, altfel maxd=d.

18. **Problema cuielor** *(Greedy optim)*

Fie N scânduri de lemn, descrise ca niste intervale închise cu capete reale. Găsiti o multime minimă de cuie astfel încât fiecare scândură să fie bătută de cel putin un cui. Se cere pozitia cuielor.

Formulat matematic: găsiti o multime de puncte de cardinal minim M astfel încât pentru orice interval [ai, bi] din cele N, să existe un punct x din M care să apartină intervalului [ai, bi].
Complexitate: O(N log N)

Exemplu:

- intrare: N = 5, intervalele: [0, 2], [1, 7], [2, 6], [5, 14], [8, 16]
- iesire: M = {2, 14}
- explicatie: punctul 2 se afla în primele 3 intervale, iar punctul 14 în ultimele 2

Solutie: Se observa că dacă x este un punct din M care nu este capăt dreapta al nici unui interval, o translatie a lui x la dreapta care îl duce în capătul dreapta cel mai apropiat nu va schimba intervalele care contin punctul. Prin urmare, exista o multime de cardinal minim M pentru care toate punctele x sunt capete dreapta.

Astfel, vom crea multimea M folosind numai capete dreapta în felul următor:

- cât timp au mai rămas intervale nemarcate:
    - selectăm cel mai mic capăt dreapta, Bmin; acesta trebuie să fie în M, deoarece este singurul punct care se afla în interiorul intervalului care se termină în Bmin
    - marcăm toate intervalele nemarcate care contin Bmin
    - adăugăm Bmin la M

Pentru a obtine o complexitate redusă, sortăm initial toate cele 2N capete si le parcurgem de la stânga la dreapta. Pentru fiecare punct distingem cazurile:

- dacă este capăt stânga, introducem intervalul în lista de „intervale în procesare” si trecem mai departe;
- dacă este capăt dreapta si intervalul respectiv nu contine nici un punct din M, atunci am găsit cel mai mic capăt dreapta al unui interval nemarcat, introducem capătul în M si marcăm toate intervalele din lista de intervale în procesare;
- dacă este capăt dreapta si intervalul din care face parte este deja marcat, trecem mai departe.

Complexitate:

- sortare: O(N log N)
- parcurgerea capetelor: O(N)
- adăugarea si stergerea unui interval din lista de intervale în procesare: O(1)
- total: O(N log N)