Temă la Analiza Algoritmilor
Problema rucsacului

Bogdan Valentin-Răzvan, grupa 321CA
valentin.bogdan3003@stud.acs.upb.ro

Facultatea de Automatică și Calculatoare
Calculatoare și Tehnologia Informației

1 Introducere

1.1 Descrierea problemei

Problema rucsacului constă în existența a N obiecte, fiecare având asociat o greutate g_i și o valoare v_i. Dintre acestea, trebuie ales un subset de obiecte, astfel încât suma greutăților să nu depășească un prag G, iar suma valorilor să fie maximă.

Face parte din grupul problemelor de optimizare, având numeroase aplicații directe în viața reală:

- organizarea coletelor unei firme de curierat într-un vehicul de transport, fără a depăși o greutate impusă, "greutatea" făcând referință chiar la greutatea reală a obiectului;
- construirea echipei unei organizații sportive profesioniste, având un buget fix, "greutatea unui sportiv" fiind suma de bani necesară semnării unui contract cu acesta;
- existența unui algoritm de criptare ce are la bază două chei, una pentru a cripta și cealaltă pentru decriptare, fiecare obținându-se rezolvând o problemă de tip rucsac.

1.2 Specificarea soluțiilor

Soluția folosind backtracking

Cea mai intuitivă soluție, dar și naivă, o reprezintă cea folosind metoda backtracking.

Se vor genera toate subseturile, iar dintre cele în care suma greutăților obiectelor este cel mult G se va alege subsetul cu suma valorilor maximă.

Spre exemplu, într-o configurație oarecare se poate atribui fiecărui obiect valoarea 0 (neales în configurația prezentă) sau 1 (ales în configurația prezentă), iar pentru a obține suma obiectelor alese se iterează prin vectorul de obiecte și se adună doar cele care au fost asignate cu 1.

Întrucât fiecare obiect poate avea una dintre cele două stări (0/1) se poate conclude că, din punct de vedere al timpului de execuție, complexitatea soluției

este exponențială, $O(2^N)$, iar complexitatea de spațiu este liniară, $O(N)$, unde N este numărul de obiecte.

Soluția folosind programare dinamică

În continuare, se poate observa că problema poate fi împărțită în subprobleme de dimensiuni mai reduse, ceea ce sugerează folosirea programării dinamice.

Mai exact, se va defini o matrice, fie $dp[i][j]$, ce reprezintă o soluție parțială (suma valorilor maximă), în care se alege un subset din primele i obiecte, având suma greutăților cel mult j. Desigur, soluția finală se găsește în $dp[N][G]$.

Având soluția parțială pentru primele $i-1$ obiecte, se poate obține și soluția adăugând obiectul i, folosind tranziția, suma greutăților fiind maxim j:

$$dp[i][j] = \begin{cases} dp[i-1][j] & \text{if } g_i > j \\ max(dp[i-1][j], dp[i-1][j-g_i] + v_i) & \text{if } g_i \leq j \end{cases}$$

Legat de implementare, se va itera prin vectorul de obiecte, iar pentru fiecare se va folosi tranziția enunțată anterior, unde j = $\overline{1, G}$

Acestea fiind spuse, complexitatea de timp este $O(N * G)$, iar cea de spațiu $O(N * G)$, unde N este numărul de obiecte, iar G greutatea maximă a obiectelor alese.

Un detaliu interesant de observat este acela că, pentru fiecare tranziție, se folosește doar linia anterioară $(i-1)$ din matricea dp, restul $i' = \overline{1, i-2}$ fiind neutilizate. Prin urmare, prima dimensiune (N) din matricea dp poate fi eliminată, optimizând complexitatea de spațiu la $O(G)$.

Din nefericire, soluția care optimizează complexitatea de spațiu face imposibilă obținerea pozițiilor obiectelor care construiesc soluția finală.

Soluția folosind greedy

Ultima soluție abordată folosește metoda greedy. În consecință, pentru fiecare obiect vom afla raportul $\frac{v_i}{g_i}$ ce reprezintă valoarea obținută pe unitatea de greutate.

Foarte important de menționat este că, spre deosebire de restul soluțiilor, aceasta nu garantează un rezultat corect. Însă, în cazul unor intrări în care raporturile $\frac{v_i}{g_i}$ sunt apropiate, metoda va avea o performanță bună.

În continuare, vom sorta descrescător obiectele pe baza raportului. Pentru a construi soluția se va itera prin obiectele sortate și vom adăuga obiectul i dacă greutatea rămasă e mai mare sau egală ca cea a obiectului actual și vom actualiza greutatea rămasă de completat.

Pentru sortare vom folosi un algoritm cu complexitatea de timp de $O(N \log N)$, ceea ce dă și complexitatea soluției, iar complexitatea de spațiu este $O(N)$, unde N este numărul de obiecte.

Rezumat

În final, am adunat toate soluțiile în tabelul de mai jos pentru o mai ușoară vizionare.

Table 1. Rezumatul soluțiilor

Metodă	Timp	Spațiu
backtracking	$O(2^N)$	$O(N)$
programare dinamică	$O(N * G)$	$O(N * G)$
greedy	$O(N \log N)$	$O(N)$

1.3 Evaluare

Soluția folosind backtracking

Întrucât complexitatea de timp a soluției de față este exponențială, compromisul acestei soluții este dat de timpul de rulare, chiar dacă va genera rezultatul corect spre deosebire de cel care folosește metoda greedy.

Prin urmare, se vor genera teste care să surprindă creșterea foarte rapidă a timpului de rulare, depășind pragul de o secundă chiar de la valori alea lui $N \approx 30$.

Soluția folosind programare dinamică

Această soluție, asemenea celei folosind backtracking, generează un rezultat corect. Plusul adus în legătură cu timpul de execuție este umbrit de folosirea mai multor resurse în materie de memorie.

În concluzie, soluția va fi testată, în principiu, având valori ale lui N și G înspre limita superioară, întrucât produsul $N * G$, din complexitățile de timp și spațiu, va deveni problematic.

Soluția folosind greedy

Din punct de vedere al complexităților, metoda greedy se descurcă cel mai bine, chiar și la limitele superioare ale N și G. Desigur, este și singura soluție care nu garantează un rezultat corect, ceea ce se va urmări în mod principal în timpul testării.

Testarea va surprinde diferențele dintre rezultatele obținute cu această metodă comparativ cu cele două menționate anterior, în încercarea de a decide dacă avantajul obținut din complexitățile reduse produce sau nu diferențe semnificative în generarea soluției.

Deoarece soluția se bazează pe compararea unor raporturi, mai exact numere reale, pot apărea erori de calcul. De asemenea, folosirea tipului de date real este

mult mai lent comparativ cu cel de date întregi.

2 Prezentarea soluțiilor

2.1 Funcționalitatea algoritmilor

Soluția folosind backtracking

Această soluție presupune generarea tuturor submulțimilor de obiecte care sunt candidate la soluție. În total, există 2^N submulțimi, iar pentru generarea acestora se va folosi o mască pe N biți, iar al i-lea bit va fi setat dacă al i-lea obiect face parte din submulțimea respectivă.

Se va itera cu un contor $i = \overline{0, 2^N - 1}$, iar pentru fiecare dintre aceste mulțimi se va verifica dacă suma greutăților obiectelor respectă condiția din enunț, în acest caz, se va compara cu cea mai bună soluție actuală și, eventual, va fi actualizată.

De observat că, odată ce suma greutăților obiectelor dintr-o submulțime depășește greutatea maximă G, submulțimea canditat este eliminată din cauza condițiilor inițiale, fără a mai lua în calcul și restul obiectelor rămase.

La afișare, se va folosi submulțimea ce respectă condiția din enunț, având și suma valorilor maximă, obținută în cadrul generării.

Soluția folosind programare dinamică

Această soluție valorifică faptul că problema poate fi împărțită în subprobleme mai mici, astfel apare posibilitatea de a folosi tehnica programării dinamice.

Se va construi o matrice de dimensiunea $N * G$, fiecare element definând o stare, formată din două caracteristici, $dp[i][j]$, reprezentând profitul maxim obținut, luând o submulțime din primele i obiecte, aceastea având greutatea maximă j. Natural, apare ideea tranziției între stări:

$$dp[i][j] = \begin{cases} dp[i-1][j] & \text{if } g_i > j \\ max(dp[i-1][j], dp[i-1][j-g_i] + v_i) & \text{if } g_i \leq j \end{cases}$$

Practic, se va itera cu un contor $i = \overline{0, N-1}$ prin mulțimea de obiecte, iar pentru fiecare cu alt contor $j = \overline{0, G}$ pentru a seta greutate maximă a submulțimii alese din cele i obiecte. Inițial, se va defini valoarea maximă a stării ca find egală cu cea care conține doar primele $i - 1$ obiecte (mai exact o subproblemă), iar, ulterior, se va verifica dacă adăugând obiectul i profitul maxim este mai mare. Verificarea se face doar în cazul când greutatea obiectului i nu depășește greutatea maximă setată j.

Pentru afișare, trebuie înțeles modul în care valorile sunt stocate în matricea dp. Așadar, pentru o stare $dp[i][j]$, valorea inițială va fi cea din $dp[i-1][j]$ și se poate modifica doar când este convenabil a introduce obiectul i în soluție. Prin

urmare, se vor căuta aceaste ”modificări” între stările care au aceeeași greutate maximă setată j în matricea dp pentru a reconstrui submulțimea ce este soluție.

Se va porni din $dp[N-1][G]$ (colțul dreapta jos al matricei), unde se află profitul maxim și se va ”urca” până când se va găsi o stare $dp[i][G]$ diferită de $dp[i-1][G]$, însemnând că obiectul i a fost introdus la soluție. Se va actualiza G, prin scăderea greutății obiectului i despre care știm că face parte din soluție și se va repeta procedeul până când se ajunge în starea inițială ($dp[0][0]$).

Pentru o mai bună înțelegere, se va considera următorul input:
6 10
3 7
3 4
1 2
1 9
2 4
1 5

Table 2. matricea dp a input-ului

i/j	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	7	7	7	7	7	7	7	7
1	0	0	0	7	7	7	11	11	11	11	11
2	0	2	2	7	9	9	11	13	13	13	13
3	0	9	11	11	16	18	18	20	22	22	22
4	0	9	11	13	16	18	20	22	22	24	26
5	0	9	14	16	18	21	23	25	27	27	29

Pașii pentru a reconstitui submulțimea soluție sunt următorii (de observat că indexarea se face de la 0):

- se pornește din starea $dp[5][10]$ care dă și profitul maxim;
- se ”urcă” pe coloana 10 până când întâlnim două stări vecine cu profitul diferit. Această diferență apare chiar între liniile 5 și 4, așadar înseamnă că al 6-lea obiect face parte din soluție (starea este notată distinctiv, culoare roșie);
- se actualizează greutatea rămasă care definește coloana pe care se va căuta în continuare; aceasta devine $10 - g(5) = 9$, iar noua linie de pe care se va începe ”urcarea” este 4. Așadar, noua căutare începe din starea $dp[4][9]$;

- se "urcă" pe coloana 9 până când întâlnim două stări vecine cu profitul diferit. Această diferență apare chiar între liniile 4 și 3, așadar înseamnă că al 5-lea obiect face parte din soluție (starea este notată distinctiv, culoare roșie);
- se actualizează greutatea rămasă care definește coloana pe care se va căuta în continuare; aceasta devine $9 - g(4) = 7$, iar noua linie de pe care se va începe "urcarea" este 3. Așadar, noua căutare începe din starea $dp[3][7]$;
- se "urcă" pe coloana 7 până când întâlnim două stări vecine cu profitul diferit. Această diferență apare chiar între liniile 3 și 2, așadar înseamnă că al 4-lea obiect face parte din soluție (starea este notată distinctiv, culoare roșie);
- se actualizează greutatea rămasă care definește coloana pe care se va căuta în continuare; aceasta devine $7 - g(3) = 6$, iar noua linie de pe care se va începe "urcarea" este 2. Așadar, noua căutare începe din starea $dp[2][6]$;
- se "urcă" pe coloana 6 până când întâlnim două stări vecine cu profitul diferit. Această diferență apare chiar între liniile 1 și 0, așadar înseamnă că al 2-lea obiect face parte din soluție (starea este notată distinctiv, culoare roșie);
- se actualizează greutatea rămasă care definește coloana pe care se va căuta în continuare; aceasta devine $6 - g(1) = 3$, iar noua linie de pe care se va începe "urcarea" este 0. Așadar, noua căutare începe din starea $dp[0][3]$;
- se "urcă" pe coloana 3 până când întâlnim două stări vecine cu profitul diferit. Fiind prima linie, iar greutatea rămasă e diferită de 0 înseamnă că și primul obiect face parte din soluție
- se actualizează greutatea rămasă care definește coloana pe care se va căuta în continuare; aceasta devine $3 - g(1) = 0$, iar noua linie de pe care se va începe "urcarea" este 0. Așadar, noua căutare începe din starea $dp[0][0]$;
- odată ce starea de început este $dp[0][0]$ înseamnă că s-au toate obiectele submulțimii soluție au fost adăugate.

La final, se afișează valoarea din $dp[N-1][G]$ și soluția reconstituită la input-ul respectiv.

Soluția folosind greedy

Această soluție folosește o abordare euristică pentru a rezolva problema. Așadar, mulțimea inițială de obiecte va fi sortată în ordine descrescătoarea în funcție de raportul $\frac{v_i}{g_i}$ întrucât se dorește obținerea unui profit cât mai mare pe unitatea de greutate. De menționat că se va păstra pentru fiecare obiect indicele în vectorul inițial, acesta fiind introdus în soluția finală.

În continuare, se va itera prin vectorul sortat de obiecte și se vor adăuga la soluție cele care au greutatea mai mică sau egală cu greutatea care mai poate fi adăugată la momentul respectiv și, în acest caz, se va actualiza greutatea

rămasă.

La final, se vor afișa obictele alese în urma iterării prin vectorul sortat și suma valorilor acestora.

2.2 Analiza complexității soluțiilor

Soluția folosind backtracking

Această soluție generează toate submulțimile de obiecte folosind o mască de biți de dimensiunea N în care fiecare obict are bitul 1 dacă apartine submulțimii sau 0 în caz contrar. Așadar, complexitatea de timp a soluției este $O(2^N)$. Referitor la resursele de memorie folosite, este nevoie doar de doi vectori adiționali pentru a reține o anumită submulțime, respectiv submulțimea soluție. Prin urmare, complexitatea de memorie este $O(N)$.

Soluția folosind programare dinamică

Această soluție construiește matricea *dp* conform tranzițiilor descrise, fiind cea mai costisitoare operație din acest algoritm, întrucât reconstituirea soluției are complexitate de timp și spațiu $O(N)$. Așadar, complexitatea finală de timp și de spațiu este $O(N * G)$.

Soluția folosind greedy

Această soluție sortează obiectele conform descrierii, folosind un algoritm de sortare de complexitate de timp $O(N \log N)$ și se iterează ulterior prin vector pentru a construi soluția. În concluzie, complexitatea de timp este $O(N \log N)$, iar cea de spațiu $O(N)$.

2.3 Principalele avantaje și dezavantaje ale soluțiilor

Soluția folosind backtracking

Avantaje:

- soluție intuitivă, nu necesită multe cunoștiințe;
- complexitate de spațiu liniară;
- oferă soluția corectă indiferent de datele de intrare.

Dezvantaje:

- complexitate de timp exponențială;
- poate fi folosită pentru un set infim de teste, în care $N < 25$.

Soluția folosind programare dinamică

Avantaje:

- complexitatea în timp și spațiu este rezonabilă, acoperind o plajă largă de teste;
- oferă soluția corectă indiferent de datele de intrare.

Dezvantaje:

- necesită înțelegerea tehnicii de programare dinamică;
- complexitatea depinde de două variabile, N și G.

Soluția folosind greedy

Avantaje:

- soluție intuitivă;
- acoperă toate posibilitățile de teste, datorită complexității optime.

Dezvantaje:

- posibile erori la operațiile de adunare/scădere/înmulțire/împărțire, mai ales în cazul valorilor reale;
- nu oferă soluția corectă în majoritatea testelor, ci doar o aproximare.

3 Evaluare

3.1 Construcția setului de teste

Testele propuse sunt împărțite în trei secțiuni, în funcție de numărul N de obiecte:

- dimensiune redusă, unde N ia valorile $3, 6, 9, ..., 24$, iar G ia valorile $30, 60, ..., 240$;
- dimensiune medie, unde N ia valorile $100, 200, ..., 900$, iar G ia valorile $100, 200, ..., 900$;
- dimensiune mare, unde N ia valorile $1000, 2000, ..., 10000$, iar G ia valorile $10000, 20000, ..., 100000$.

În plus, pentru fiecare pereche de N și G se generează trei tipuri de teste, în funcție de împrăștiere valorilor raporturilor $\frac{v_i}{g_i}$:

- împrăștiere redusă, astfel încât fiecare obiect este de tipul (g, v), unde g reprezintă greutatea, iar v valoarea, între ele fiind egalitatea $g = v + 1$, g fiind generat cu ajutorul srand() (exemplu: (10, 9); (100, 99));
- împrăștiere medie, astfel încât fiecare obiect este de tipul (g, v), unde g reprezintă greutatea, iar v valoarea, g fiind generat cu ajutorul srand(), iar v fiind generat în funcție de g, respectând inegalitatea $2 \leq v \leq g$;

– împrăștiere mare, astfel încât fiecare obiect este de tipul (g, v), unde g reprezintă greutatea, iar v valoarea, g fiind generat cu ajutorul srand(), iar v fiind generat în funcție de g, respectând inegalitatea $1 \leq v \leq g$.

De observat că toate raporturile $\frac{v_i}{g_i} \leq 1$ pentru a controla seturile de teste generate și a evidenția comportamentul fiecărui algoritm.

3.2 Specificațiile sistemului de calcul

Procesor:

– nume: Intel Core i5 7300HQ;
– frecvență: 2.50GHz;
– tehnologie: 14nm;
– nuclee: 4, threaduri: 4.

Memorie:

– tip: DDR4;
– dimensiune: 8GB;
– frecvență: 2400MH;
– canal: single.

Compilator:

– nume: C++;
– versiune: 11.3.0;
– flag extra: pragma optimize O3.

3.3 Ilustrarea rezultatelor evaluării soluțiilor

Rezultate referitoare la timpul de rulare al soluțiilor

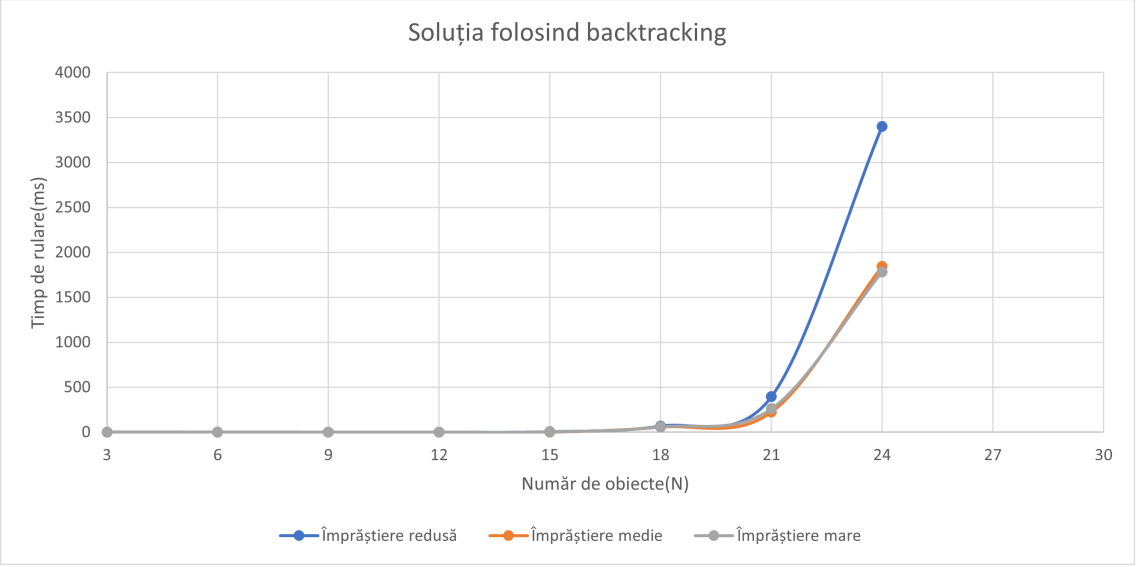

Fig. 1. Rezultatele obținute pe testele de dimensiune redusă

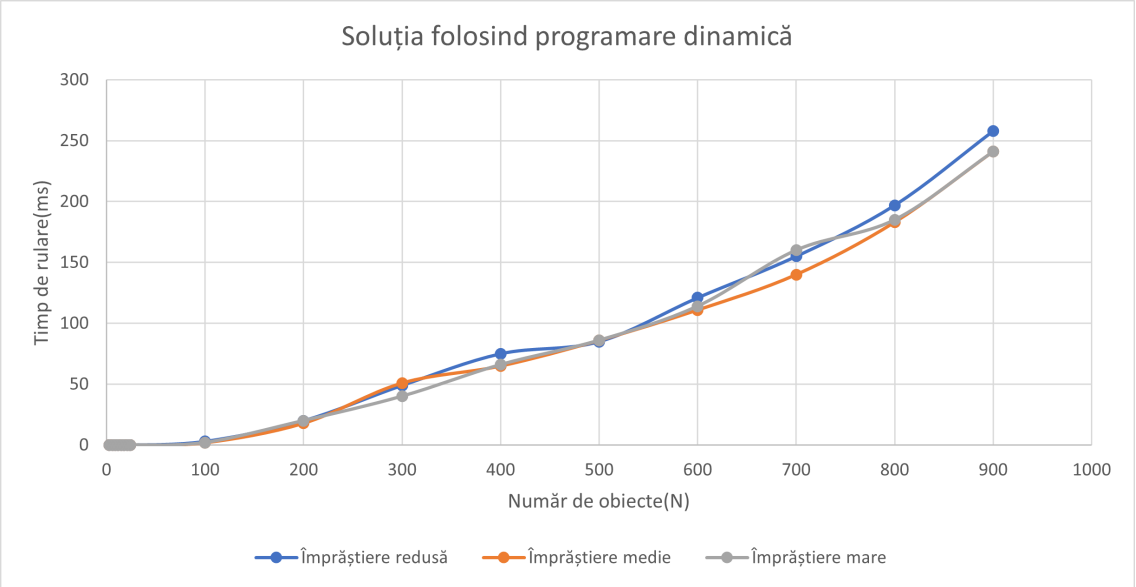

Fig. 2. Rezultatele obținute pe testele de dimensiune redusă și medie

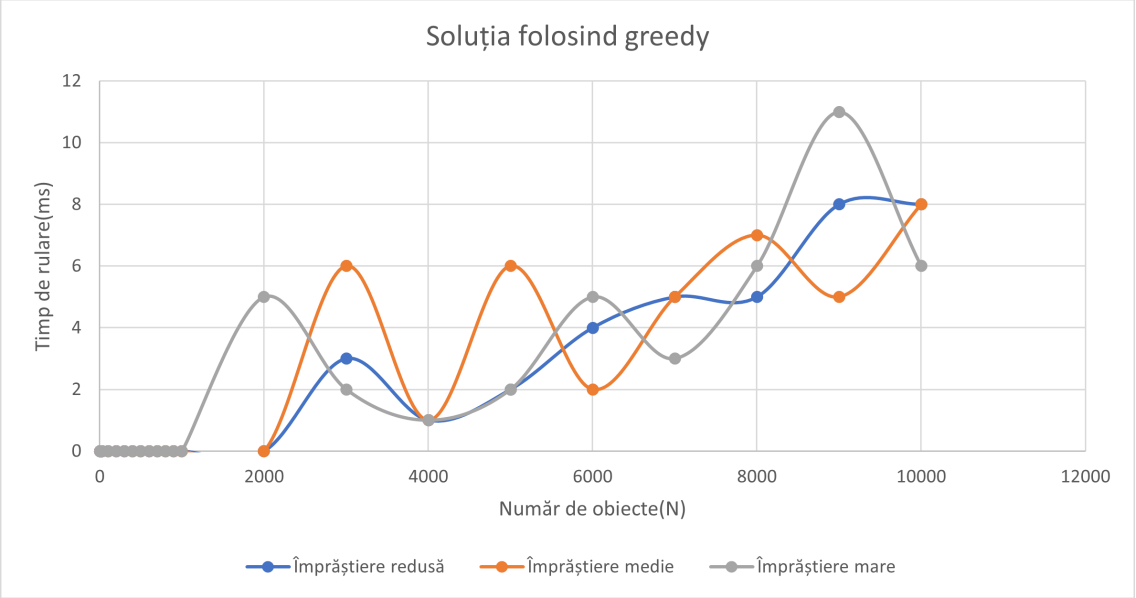

Fig. 3. Rezultatele obținute pe testele de dimensiune redusă, medie și mare

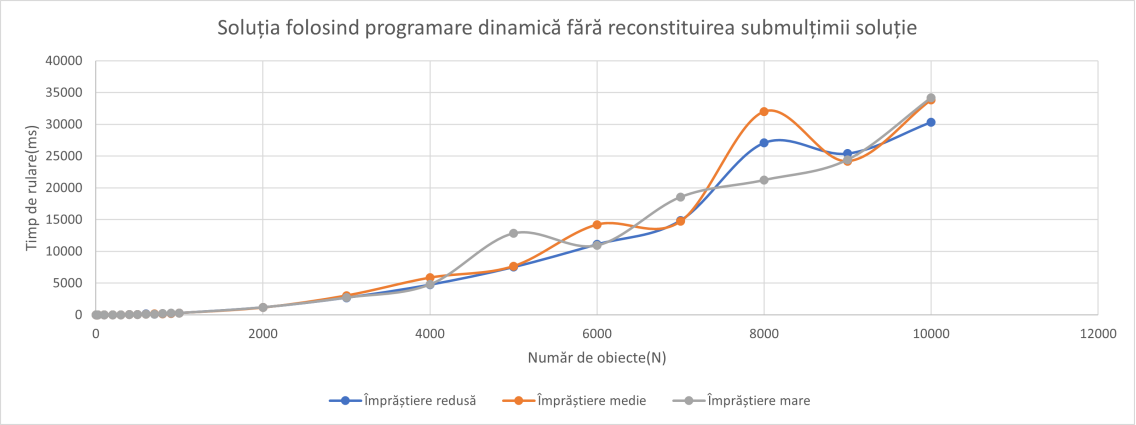

Fig. 4. Rezultatele obținute pe testele de dimensiune redusă, medie și mare

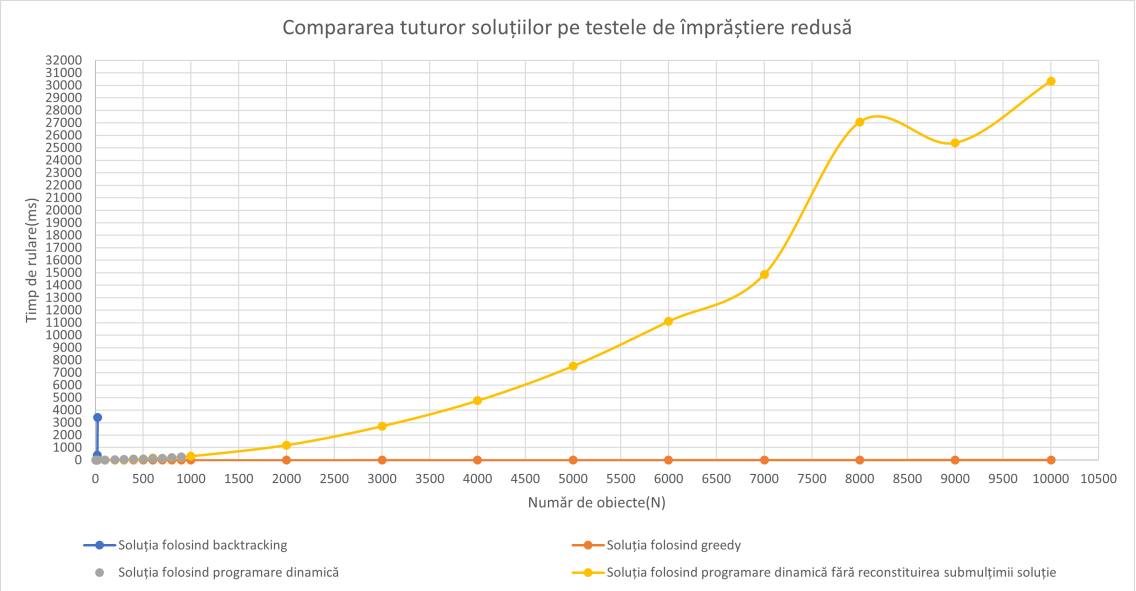

Fig. 5. Rezultatele obținute pe toate testele de împrăștiere redusă

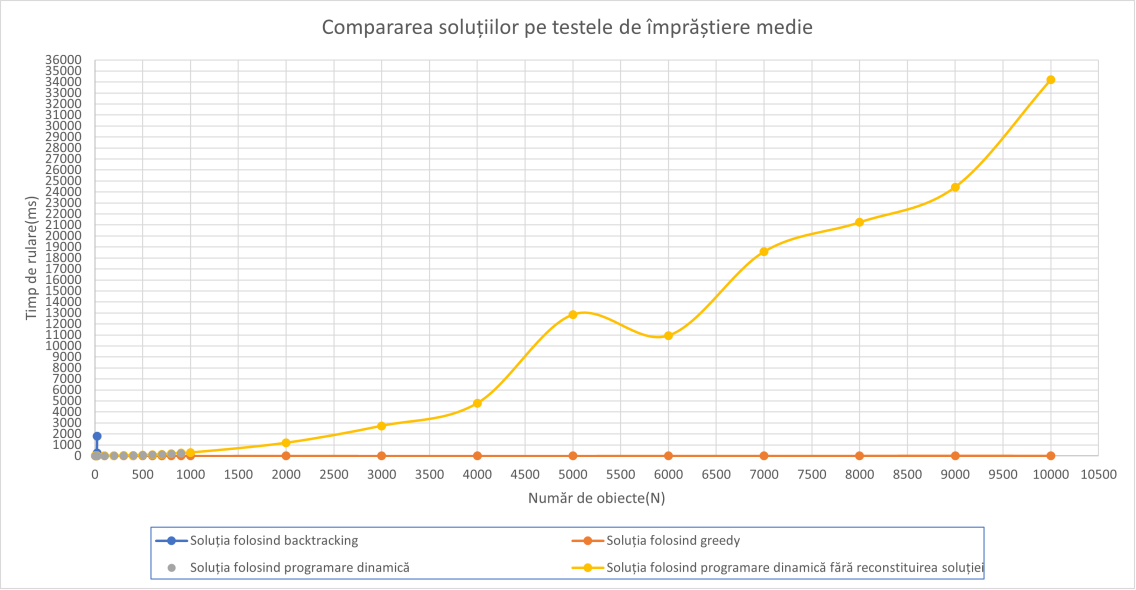

Fig. 6. Rezultatele obținute pe toate testele de împrăștiere medie

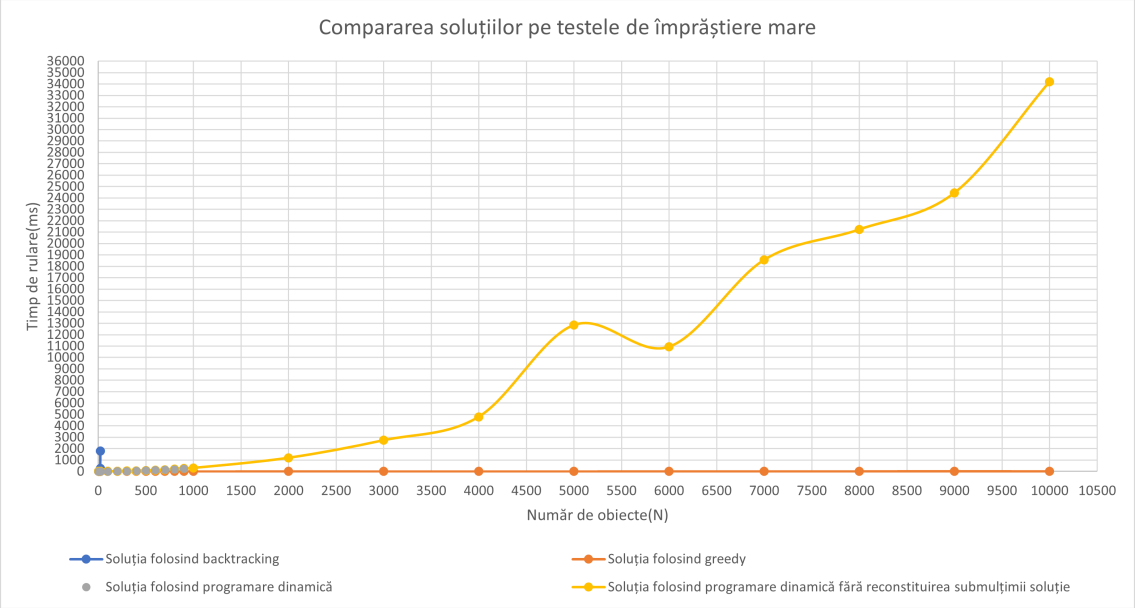

Fig. 7. Rezultatele obținute pe toate testele de împrăștiere mare

Rezultatele obținute sugerează următoarele:

- soluția folosind metoda greedy este, pe departe, cea mai optimă din punct de vedere al timpului de rulare;
- diferența dintre gradul de împrăștiere al datelor influențează numai metoda backtracking, deoarece condiția de oprire specifică acestei metode este rar apelată din cauza modului de contrucție al testelor ($g = v + 1$);
- cele două soluții folosind programare dinamică se comportă la fel întrucât diferența dintre cele două nu reprezintă elementul de complexitate majoritar (reconstituirea soluției: $O(N)$);
- complexitatea de timp exponențială a soluției folosind backtracking permite doar rularea pe testele de dimensiune redusă.

Rezultate referitoare la aproximarea soluției corecte

Table 3. Rezultatele obținute pe testele de dimensiuni mici

Soluția folosind backtracking					
grad de împrăștiere	N	G	soluție aproximativă	soluție corectă	eroare
redus	3	30	6	6	0
	6	60	21	21	0
	9	90	41	41	0
	12	120	64	41	0
	15	150	100	41	0
	18	180	143	143	0
	21	210	195	195	0
	24	240	226	226	0
mediu	3	30	30	30	0
	6	60	60	60	0
	9	90	90	90	0
	12	120	120	120	0
	15	150	150	150	0
	18	180	180	180	0
	21	210	210	210	0
	24	240	240	240	0
mare	3	30	30	30	0
	6	60	60	60	0
	9	90	90	90	0
	12	120	120	120	0
	15	150	150	150	0
	18	180	180	180	0
	21	210	210	210	0
	24	240	240	240	0

Table 4. Rezultatele obținute pe toate testele

Soluția folosind programare dinamică fără reconstituirea submulțimii soluție					
grad de împrăștiere	N	G	soluție aproximativă	soluție corectă	eroare
redus	3	30	6	6	0
	6	60	21	21	0
	9	90	41	41	0
	12	120	64	64	0
	15	150	100	100	0
	18	180	143	143	0
	21	210	195	195	0
	24	240	226	226	0
	100	1000	989	989	0
	200	2000	1989	1989	0
	300	3000	2989	2989	0
	400	4000	3989	3989	0
	500	5000	4989	4989	0
	600	6000	5989	5989	0
	700	7000	6989	6989	0
	800	8000	7989	7989	0
	900	9000	8989	8989	0
	1000	10000	9989	9989	0
	2000	20000	19989	19989	0
	3000	30000	29989	29989	0
	4000	40000	39989	39989	0
	5000	50000	49989	49989	0
	6000	60000	59989	59989	0
	7000	70000	69989	69989	0
	8000	80000	79989	79989	0
	9000	90000	89989	89989	0
	10000	100000	99989	99989	0
mediu	3	30	30	30	0
	6	60	60	60	0
	9	90	90	90	0
	12	120	120	120	0
	15	150	150	150	0
	18	180	180	180	0
	21	210	210	210	0
	24	240	240	240	0
	100	1000	1000	1000	0
	200	2000	2000	2000	0
	300	3000	3000	3000	0
	400	4000	4000	4000	0
	500	5000	5000	5000	0
	600	6000	6000	6000	0
	700	7000	7000	7000	0
	800	8000	8000	8000	0
	900	9000	9000	9000	0
	1000	10000	10000	10000	0
	2000	20000	20000	20000	0
	3000	30000	30000	30000	0
	4000	40000	40000	40000	0
	5000	50000	50000	50000	0
	6000	60000	60000	60000	0
	7000	70000	70000	70000	0
	8000	80000	80000	80000	0
	9000	90000	90000	90000	0
	10000	100000	100000	100000	0

Table 5. Rezultatele obținute pe toate testele

Soluția folosind programare dinamică fără reconstituirea submulțimii soluție					
grad de împrăștiere	N	G	soluție aproximativă	soluție corectă	eroare
mare	3	30	30	30	0
	6	60	60	60	0
	9	90	90	90	0
	12	120	120	120	0
	15	150	150	150	0
	18	180	180	180	0
	21	210	210	210	0
	24	240	240	240	0
	100	1000	1000	1000	0
	200	2000	2000	2000	0
	300	3000	3000	3000	0
	400	4000	4000	4000	0
	500	5000	5000	5000	0
	600	6000	6000	6000	0
	700	7000	7000	7000	0
	800	8000	8000	8000	0
	900	9000	9000	9000	0
	1000	10000	10000	10000	0
	2000	20000	20000	20000	0
	3000	30000	30000	30000	0
	4000	40000	40000	40000	0
	5000	50000	50000	50000	0
	6000	60000	60000	60000	0
	7000	70000	70000	70000	0
	8000	80000	80000	80000	0
	9000	90000	90000	90000	0
	10000	100000	100000	100000	0

Table 6. Rezultatele obținute pe toate testele

Soluția folosind greedy					
grad de împrăștiere	N	G	soluție aproximativă	soluție corectă	eroare
redus	3	30	6	6	0
	6	60	21	21	0
	9	90	41	41	0
	12	120	64	64	0
	15	150	100	100	0
	18	180	143	143	0
	21	210	195	195	0
	24	240	225	226	1
	100	1000	989	989	0
	200	2000	1989	1989	0
	300	3000	2988	2989	1
	400	4000	3988	3989	1
	500	5000	4988	4989	1
	600	6000	5988	5989	1
	700	7000	6988	6989	1
	800	8000	7988	7989	1
	900	9000	8988	8989	1
	1000	10000	9989	9989	0
	2000	20000	19988	19989	1
	3000	30000	29989	29989	0
	4000	40000	39989	39989	0
	5000	50000	49989	49989	0
	6000	60000	59988	59989	1
	7000	70000	69989	69989	0
	8000	80000	79988	79989	1
	9000	90000	89989	89989	0
	10000	100000	99988	99989	1
mediu	3	30	30	30	0
	6	60	17	60	43
	9	90	41	90	46
	12	120	62	120	58
	15	150	98	150	52
	18	180	131	180	49
	21	210	152	210	58
	24	240	155	240	85
	100	1000	894	1000	106
	200	2000	1893	2000	107
	300	3000	2877	3000	123
	400	4000	4000	4000	0
	500	5000	5000	5000	0
	600	6000	5795	6000	205
	700	7000	6886	7000	114
	800	8000	8000	8000	0
	900	9000	8887	9000	113
	1000	10000	9880	10000	120
	2000	20000	19845	20000	155
	3000	30000	29859	30000	141
	4000	40000	40000	40000	0
	5000	50000	50000	50000	0
	6000	60000	60000	60000	0
	7000	70000	69762	70000	238
	8000	80000	80000	80000	0
	9000	90000	89877	90000	123
	10000	100000	99730	100000	270

Table 7. Rezultatele obținute pe toate testele

Soluția folosind greedy					
grad de împrăștiere	N	G	soluție aproximativă	soluție corectă	eroare
mare	3	30	7	30	23
	6	60	16	60	44
	9	90	36	90	54
	12	120	56	120	64
	15	150	88	150	62
	18	180	96	180	84
	21	210	151	210	59
	24	240	148	240	92
	100	1000	959	1000	41
	200	2000	1920	2000	80
	300	3000	2940	3000	60
	400	4000	3939	4000	61
	500	5000	4960	5000	40
	600	6000	5914	6000	86
	700	7000	6930	7000	70
	800	8000	7863	8000	137
	900	9000	8928	9000	72
	1000	10000	9894	10000	106
	2000	20000	19955	20000	45
	3000	30000	29963	30000	37
	4000	40000	39900	40000	100
	5000	50000	49931	50000	69
	6000	60000	59887	60000	113
	7000	70000	69892	70000	108
	8000	80000	79919	80000	81
	9000	90000	89859	90000	141
	10000	100000	99924	100000	76

Rezultatele obținute sugerează următoarele:

- toate soluțiile exceptând cea folosind metoda greedy generează răspunsuri corecte, indiferent de datele de intrare;
- generarea datelor de intrare astfel încât gradul de împrăștiere al raporturilor $\frac{v_i}{g_i}$ să se afle într-un interval restrâns pare să dea roade, întrucât eroare maximă pentru metoda greedy este 1;
- diferența dintre erorile pentru cele două grade de împrăștiere(medii și mari) nu este semnificativă, totuși, în prima variantă, apar și rezultate care aproximează corect soluția.

4 Concluzii

În urma analizei problemei propuse, toate cele patru soluții (incluzând și cea de-a doua variantă pentru metoda programării dinamice care doar află soluția corectă, fără a reconstitui submulțimea soluție) pot reprezenta o variantă inspirată în diferite situații.

Departajarea între aceste soluții o face punerea în balanță a două elemente esențiale: precizia și timpul stabilit pentru realizare. Așadar, în cazul unui incendiu, pentru salvarea lucrurilor de valoarea trebuie aplicată metoda greedy, pe când planificarea unei călătorii pe baza unui buget se poate face aplicând metoda programării dinamice.

5 Bibliografie

Thomas H. Cormen: Introduction to Algorithms. 3rd edition.
Brilliant, https://brilliant.org/wiki/backpack-problem/. Last accessed 24 Nov 2022