Lucrare de laborator

Teodora Peanca
Grupa 3, Anul I, Secțiunea Calculatoare

Cuprins

1 Enunțul Problemei

Scopul acestei lucrări de laborator este de a rezolva următoarea problemă: un pescar trebuie să aleagă dintr-un set de homari pe aceia care au suma valorii lor maximă, dar cu suma dimensiunilor mai mică decât capacitatea plasei sale. Problema diferă de clasică problemă a rucsacului prin faptul că homarii aleși trebuie enumerați.

În mod specific, se consideră că pescarul dispune de o plasă cu o capacitate maximă dată. El are la dispoziție un număr nedeterminat de homari, fiecare având asociate trei caracteristici esențiale: nume, dimensiune și valoare. Programul trebuie să primească ca date de intrare capacitatea maximă a plasei și informații detaliate despre fiecare homar în parte. Obiectivul este de a determina combinația optimă de homari care să maximizeze valoarea totală, respectând în același timp constrângerea de capacitate a plasei.

Formularea problemei poate fi prezentată matematic astfel:

- **Date de intrare**:
  - C - capacitatea maximă a plasei (o constantă pozitivă).
  - n - numărul de homari disponibili.
  - Pentru fiecare homar i ($i = 1, 2, \ldots, n$):
    * nume_i - numele homarului i (un șir de caractere).
    * dimensiune_i - dimensiunea homarului i (un număr real pozitiv).
    * valoare_i - valoarea homarului i (un număr real pozitiv).
- **Date de ieșire**:
  - Un subset de homari selectați astfel încât:
    * Suma dimensiunilor homarilor selectați să fie mai mică sau egală cu C.
    * Suma valorilor homarilor selectați să fie maximă.
    * Homarii selectați să fie enumerați.

Astfel, problema se încadrează în clasa problemelor de optimizare combinatorie, fiind o variație a problemei rucsacului, dar cu o cerință suplimentară de a enumera homarii selectați. Implementarea unui algoritm eficient pentru această problemă va trebui să țină cont de complexitatea combinatorie, asigurându-se totodată că soluția obținută este optimală în raport cu constrângerile date.

2 Algoritmi

2.1 Algoritm rucsac

Pentru aflarea valorii maxime din plasa s-a folist algoritmul de programare dinamica pentru problema rucsacului. Complexitatea de timp si memorie este de O(numar homari * capacitate plasa).

Algorithm 1 Algoritmul Rucsacului (0-1 Knapsack)

```
Definește un tabel K[0..n][0..C]
for i = 0 to n do
    for cap = 0 to C do
        if i == 0 or cap == 0 then
            K[i][cap] = 0
        else if w[i] <= cap then
            K[i][cap] = max(v[i] + K[i − 1][cap − w[i]], K[i − 1][cap])
        else
            K[i][cap] = K[i − 1][cap]
        end if
    end for
end for
return K[n][C]
```

2.2 Explicație algoritm rucsac

Să detaliem pașii algoritmului:

- Definim un tabel K unde $K[i][cap]$ reprezintă valoarea maximă care poate fi obținută utilizând primele i obiecte și având capacitatea cap disponibilă în rucsac.
- Inițializăm tabelul K cu zero pentru cazurile de bază unde fie numărul de obiecte este zero, fie capacitatea rucsacului este zero.
- Pentru fiecare obiect i și fiecare capacitate cap:
  - Dacă dimensiunea obiectului i este mai mică sau egală cu capacitatea cap, atunci avem două opțiuni:
    - Includem obiectul i și adăugăm valoarea lui la valoarea optimă a rucsacului cu capacitatea rămasă $cap - w[i]$.

    * Nu includem obiectul i și luăm valoarea optimă a rucsacului fără acest obiect.
  - Alegem opțiunea care oferă valoarea maximă.
  - Dacă dimensiunea obiectului i este mai mare decât capacitatea *cap*, nu putem include obiectul, deci păstrăm valoarea optimă fără acest obiect.
- Rezultatul final, adică valoarea maximă care poate fi obținută cu capacitatea C utilizând toate obiectele, se găsește în $K[n][C]$.

2.3 Algoritmul de enumerare a elementelor selectate

Pentru a determina care obiecte au fost selectate pentru a obține valoarea maximă, putem parcurge tabelul K în sens invers, pornind de la $K[n][C]$:

Algorithm 2 Algoritmul de Enumerare a Elementelor Selectate

```
Initializează lista goală  selectedItems
cap = C
for i = n to 1  cu pasul  -1 do
    if K[i][cap] != K[i-1][cap] then
        Adaugă obiectul  i  în  selectedItems
        cap = cap − w[i]
    end if
end for
return selectedItems
```

2.4 Explicație algoritm de enumerare a elementelor selectate

Algoritmul de enumerare a elementelor selectate utilizează tabelul K pentru a identifica obiectele incluse în soluția optimă:

- Începem de la poziția $K[n][C]$ și verificăm dacă valoarea curentă diferă de cea de la poziția de deasupra ei $K[i-1][cap]$.
- Dacă valorile sunt diferite, înseamnă că obiectul i a fost inclus în rucsac. Adăugăm obiectul în lista de obiecte selectate și reducem capacitatea disponibilă *cap* cu dimensiunea obiectului $i-1$.
- Continuăm acest proces până ajungem la primul obiect.

- Lista rezultată, selectedItems, conține obiectele care au fost incluse pentru a obține valoarea maximă.

Complexitatea algoritmului de enumerare este de O(numar obiecte), deci nu afecteaza complexitatea programului general.

3 Date Experimentale

Datele experimentale au fost generate cu ajutorul programului de generare de teste. A rezultat urmatorul tabel:

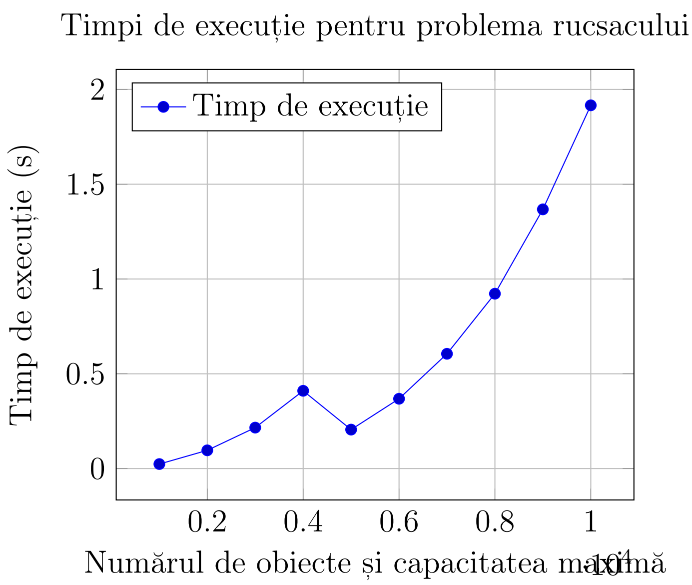

Figure 1: Graficul timpilor de execuție pentru problema rucsacului

3.1 Interpretare

Analizând graficul timpilor de execuție pentru problema rucsacului, putem trage următoarele concluzii:

- Timpii de execuție cresc în general odată cu creșterea numărului de obiecte și a capacității maxime a rucsacului. Aceasta este o așteptare normală, deoarece algoritmul de programare dinamică folosit are o complexitate de $O(n * C)$, unde n este numărul de obiecte și C este capacitatea maximă a rucsacului.

- Observăm că până la testul 5, timpii de execuție cresc relativ liniar, dar începând cu testul 6, există o schimbare în tendința de creștere a timpului de execuție. Acest lucru poate fi explicat prin schimbarea intervalului valorilor și dimensiunilor obiectelor de la 1-1000 la 1-10000, care ar putea introduce o variabilitate suplimentară în calcul.
- În testele de la 6 la 10, timpii de execuție cresc semnificativ, sugerând că intervalul mai mare al valorilor și dimensiunilor obiectelor influențează performanța algoritmului.
- Testul 5 prezintă un timp de execuție mai mic decât testul 4, ceea ce poate indica o variabilitate în datele de intrare care permite o soluție mai rapidă în acel caz specific. Aceasta arată că în practică, chiar și pentru probleme similare, performanța poate varia în funcție de particularitățile setului de date.

4 Rezultate și Concluzii

Graficul timpilor de execuție pentru algoritmul rucsacului evidențiază câteva aspecte importante legate de performanța și eficiența acestuia în funcție de numărul de obiecte și capacitatea maximă a rucsacului:

- **Creșterea timpului de execuție în funcție de dimensiunea problemei:** Timpii de execuție cresc în general odată cu creșterea numărului de obiecte și a capacității maxime a rucsacului. Aceasta reflectă complexitatea algoritmului, care este O(n×C), unde n este numărul de obiecte și C este capacitatea rucsacului.
- **Impactul intervalului valorilor și dimensiunilor obiectelor:** Observăm o schimbare semnificativă în timpii de execuție după testul 5, când intervalul valorilor și dimensiunilor obiectelor s-a schimbat de la 1-1000 la 1-10000. Aceasta sugerează că un interval mai mare de valori și dimensiuni poate introduce o variabilitate suplimentară care afectează performanța.
- **Variabilitatea datelor de intrare:** Timpul de execuție mai mic pentru testul 5 comparativ cu testul 4 indică faptul că particularitățile setului de date pot influența semnificativ performanța. Deși în medie timpii cresc, există situații în care algoritmul poate funcționa mai eficient pentru anumite date.

- **Scalabilitatea algoritmului:** Pe măsură ce dimensiunea problemei crește, timpii de execuție devin semnificativ mai mari. Aceasta subliniază importanța optimizării și a posibilei utilizări a tehnicilor de reducere a dimensiunii problemei sau de paralelizare pentru a gestiona probleme de foarte mari dimensiuni.

În concluzie, deși algoritmul de programare dinamică pentru problema rucsacului este eficient și asigură o soluție optimă, performanța sa este sensibilă la dimensiunea problemei și la caracteristicile specific ale datelor de intrare. Pentru aplicații practice, este esențială o evaluare atentă a acestor factori și, acolo unde este posibil, implementarea unor optimizări suplimentare.