Tema 1 - Proiectarea Algoritmilor

Bogdan Alexandra-Lăcrămioara, 325CD

20 aprilie 2023

Keywords: Divide et Impera · MinMax · Greedy · Egyptian Fraction · Programare dinamică · Cutting a rod · Backtracking · Fazan

1 Divide et Impera - Problema MinMax

1.1 Enunț

Se citeste n, apoi n numere intregi.
Calculaţi valoarea minimă minim și valoarea maximă maxim a celor n numere date.

1.2 Date de intrare

Fişierul de intrare "minmax.in" conţine pe prima linie numărul n si pe a doua linie n numere naturale separate prin spaţii

1.3 Date de ieșire

Fişierul de ieşire "minmax.out" va conţine pe prima linie cele două numere, minim şi maxim, separate printr-un singur spaţiu.

1.4 Input

8
10 23 5 7 25 2 3 1

1.5 Output

1 25

1.6 Pseudocod

```
minmax(v, st, dr, min, max)
        if(st == dr)  //impera
            min = max = v[st]
        end if
        else
            a_min, a_max, b_min, b_max
            m = (st + dr)/2
            minmax(v, st, m, a_min, a_max) //divide
            minmax(v, m + 1, dr,  b_min, b_max) //divide

            min = min(a_min, b_min) //combine
            max = max(a_max, b_max) //combine
        end else
end minmax
```

1.7 Descrierea soluției

În ansamblu, algoritmul "minmax" folosește trei pași(impera, divide, combine) pentru a găsi valorile minime și maxime ale unui vector de numere întregi, împărțind intervalul în jumătăți, căutând valorile minime și maxime în mod recursiv și combinându-le în final pentru a obține rezultatele dorite.

1. Dacă indicele din stânga și cel din dreapta sunt egale (impera), atunci min și max vor fi egale cu valoarea vectorului de la acel indice.
2. În caz contrar (împarte):
   - Se calculează mijlocul intervalului (m = (st + dr) / 2).
   - Se apelează recursiv funcția minmax pentru jumătatea din stânga a intervalului și jumătatea din dreapta a intervalului.
   - Se obțin valorile minime și maxime pentru cele două jumătăți (a_min, a_max, b_min, b_max).

3. Se combină valorile (combine):
   - Minimul va fi minimul dintre a_min și b_min.
   - Maximul va fi maximul dintre a_max și b_max.

4. Se returnează valorile min și max.

În concluzie, algoritmul "minmax" împarte intervalul de căutare în jumătăți și calculează valorile minime și maxime pentru fiecare jumătate în mod recursiv, iar apoi combină valorile obținute pentru a obține valorile minime și maxime finale ale întregului interval.
Algoritmul prezentat folosește o metodă top-down.

1.8 Rularea pas cu pas a algoritmului pe inputul dat:

```
v = {10, 23, 5, 7, 25, 2, 3, 1 }

min = 0, max = 0

st = 0, dr = 7

1. Se apelează minmax(v, 0, 7, 0 , 0)

/* Schema apelurilor recursive */

minmax(v, 0, 3, &a_min, &a_max)
    minmax(v, 0, 1, &a_min, &a_max)
        minmax(v, 0, 0, &a_min, &a_max) => min = 10
        minmax(v, 1, 1, &b_min, &b_max) => max = 23
                                                    => min = 5 si max = 23 (1)
     minmax(v, 2, 3, &a_min, &a_max)
        minmax(v, 2, 2, &a_min, &a_max) => min = 5
        minmax(v, 3, 3, &b_min, &b_max) => max = 7

minmax(v, 4, 7, &a_min, &a_max)
    minmax(v, 4, 5, &a_min, &a_max)
        minmax(v, 4, 4, &a_min, &a_max) => min = 2
        minmax(v, 5, 5, &b_min, &b_max) => max = 25
                                                    => min = 1 si max = 25 (2)
    minmax(v, 6, 7, &a_min, &a_max)
        minmax(v, 6, 6, &a_min, &a_max) => min = 1
        minmax(v, 7, 7, &b_min, &b_max) => max = 3

Din(1) si (2) => min = 1
                 max = 25
```

Explicație:

1. La început, apelăm funcția minmax cu vectorul v, și intervalul [0, 7], și inițializăm valorile pentru min și max cu 0.
2. Verificăm dacă st (începutul intervalului) este egal cu dr (sfârșitul intervalului). În acest caz, acestea nu sunt egale, așa că trecem la ramura else.
3. Calculăm mijlocul intervalului, folosind formula m = (st + dr) / 2, care în acest caz devine m = (0 + 7) / 2 = 3.
4. Facem două apeluri recursive ale funcției minmax pentru a calcula valorile minime și maxime ale vectorului v în două subintervale, [0, 3] și [4, 7].
5. Pentru subintervalul [0, 3], facem un alt apel recursiv al funcției minmax pentru a calcula valorile minime și maxime. Pentru [0, 1], găsim min = 5 și max = 23 prin apelurile recursive. Pentru [2, 3], găsim min = 5 și max = 7 prin apelurile recursive.
6. Pentru subintervalul [4, 7], facem un alt apel recursiv al funcției minmax pentru a calcula valorile minime și maxime. Pentru [4, 5], găsim min = 2 și max = 25 prin apelurile recursive. Pentru [6, 7], găsim min = 1 și max = 3 prin apelurile recursive.
7. La final, combinăm valorile minime și maxime calculate pentru cele două subintervale, obținând min = 1 și max = 25 ca valorile minime și maxime ale vectorului v în intervalul [0, 7].

1.9 Complexitate

* **Complexitate temporală:** O(n)

- Se deduce din recurența T(n) =

- $T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + T\left(\left\lceil \frac{n}{2} \right\rceil\right) + 2$, pentru $n > 2$
- 1 , pentru n = 2
- 0 , pentru n = 1

* **Complexitate spațială:** O(log(n))

2 Greedy - Egyptian Fraction

2.1 Enunț

Dat fiind un număr pozitiv rațional în formă de fracție proprie de forma unde a și b sunt numere întregi pozitive și cel mai mare divizor comun al lor este 1, problema fracțiilor egiptene constă în a găsi o serie finită de fracții unitare

$$\sum_{i=1}^{n} \frac{1}{x_i}$$

,unde xi sunt numere întregi pozitive, astfel încât suma acestor fracții să fie egală cu a/b.

2.2 Date de intrare

Programul citește din fișierul „ef.in", pe prima linie, numărul a, iar numărul b.

2.3 Date de ieșire

Programul va afișa în fișierul „ef.out", reprezentarea Egyptian Fraction a numărului de forma a/b.

2.4 Input

6 14

2.5 Output

Egyptian Fraction representation of 6/14 is 1/3 + 1/11 + 1/231

2.6 Pseudocod

```
Funcție egyptianFraction(n, d)
                    if d == 0 sau n == 0 atunci
                            Returnează
                    end if
                    if d % n == 0 atunci
                            Afișează "1/" concatenat cu d / n
                                    Returnează
                                                end if
                                                if n % d == 0 atunci
                                                        Afișează n / d
                            Returnează
                      end if

            if  n > d atunci
                       Afișează n / d concatenat cu " + "
                       egyptianFraction(n % d, d)
                       Returnează
                end if
                x ← d / n + 1
                Afișează "1/" concatenat cu x concatenat cu " + "
             egyptianFraction(n * x - d, d * x)

end egyptianFraction
```

2.7 Descrierea soluției

Definiție:

Fracție egipteană: Fracția egipteană, cunoscută și sub numele de fracție unitară, reprezintă o fracție cu numărătorul 1 si cu un numitor pozitiv, care este descompusă într-o sumă de fracții cu numărătorii 1 si cu numitori distincți si pozitivi.

1. **Pasul 1: Verificare d si n.** Algoritmul începe prin a verifica dacă d este egal cu 0 sau n este egal cu 0. Dacă una dintre aceste condiții este îndeplinită, algoritmul se oprește și nu afișează nimic, deoarece nu putem calcula fracții în care numitorul este 0 și nu avem nevoie de fracții egiptene pentru numărul 0.
2. **Pasul 2: Fracție unitară.**
   Dacă n/d poate fi exprimată ca o fracție unitară (adică d este divizibil cu n), atunci algoritmul va afișa "1/" urmat de câtul dintre d și n și se va opri. Acest lucru se face pentru a simplifica fracția, deoarece o fracție unitară este deja o fracție egipteană.
3. **Pasul 3: Fracție întreagă.**
   Dacă n/d poate fi exprimată ca un număr întreg (adică n este divizibil cu d), atunci algoritmul va afișa numărul întreg n/d și se va opri. Acest lucru se face pentru a afișa direct rezultatul ca un număr întreg, deoarece nu este necesar să îl exprimăm ca fracție egipteană dacă este deja un număr întreg.

4. **Pasul 4: n mai mare ca d.**
   Dacă n este mai mare decât d, atunci algoritmul va afișa câtul dintre n și d, urmat de "+", și apoi va apela funcția recursiv cu $n\%d$ și d, adică cu restul împărțirii lui n la d și cu d. Acest pas se repetă până când n devine mai mic sau egal cu d. Acesta este procesul de descompunere a fracției în fracții egiptene, unde afișăm câtul și restul împărțirii, și continuăm acest proces până când restul devine mai mic sau egal cu numitorul.
5. **Pasul 5: n mai mic ca d.**
   În cazul în care n este mai mic decât d, atunci algoritmul va calcula $x = \frac{d}{n} + 1$ și va afișa "1/" urmat de x și "+". Apoi, va apela funcția recursiv cu $nx - d$ și dx. Acest pas se repetă până când n devine mai mic sau egal cu d. Acesta este procesul de descompunere a fracției în fracții egiptene, unde înlocuim
6. **Pasul 6: Returneaza rultatul**
   În final, funcția va returna rezultatul calculat, care va fi o concatenare a fracțiilor egiptene obținute prin pașii anteriori.

2.8 Rularea pas cu pas a algoritmului pe inputul dat:

Pentru inputul 6/14, algoritmul va fi rulat astfel:

1. n = 6, d = 14
2. d % n = 14 % 6 = 2 , care nu este egal cu 0, deci se trece la următoarea condiție
3. n % d = 6 % 14 = 6, care nu este egal cu 0, deci se trece la următoarea condiție
4. Verifiare dacă n mai mare ca d. 6 mai mare ca 14? Nu, deci se trece la următoarea condiție
5. x = d / n + 1 = 14 / 6 + 1 = 2 + 1 = 3
6. Se afișează "1/3 + "
7. Se apelează recursiv egyptianFraction(n * x - d, d * x), adică egyptianFraction(6 * 3 - 14, 14 * 3)

Pasul recursiv:

1. n = 6 * 3 - 14 = 4, d = 14 * 3 = 42
2. d% n = 42 % 8 = 2, care nu este egal cu 0, deci se trece la următoarea condiție
3. n % d = 8 % 42 = 8, care nu este egal cu 0, deci se trece la următoarea condiție
4. Verifiare dacă n mai mare ca d. 8 mai mare ca 42? Nu, deci se trece la următoarea condiție
5. x = d / n + 1 = 42 / 4 + 1 = 10 + 1 = 11
6. Se afișează "1/11 + "
7. Se apelează recursiv egyptianFraction(n * x - d, d * x), adică egyptianFraction(4 * 11 - 42, 42 * 11)

Pasul recursiv:

1. n = 4 * 11 - 42 = 2, d = 42 * 11 = 462
2. d % n = 294 % 14 = 0, deci se realizează următoarele:
3. Se afișează "1/231"
4. Funcția se încheie, deoarece n și d sunt acum divizibile și se afișează fracția egipteană completă: "1/3 + 1/11 + 1/231".

2.9 Complexitate

* **Complexitate temporală:** O(n^2)

* **Complexitate spațială:** O(n)

2.10 Concluzii

În concluzie, acest algoritm își are aplicabilitatea și în viața de zi cu zi, spre exemplu, atunci când trebuie să împărțim o cantitate limitată de resurse între mai multe persoane în mod echitabil.

3 Programare dinamică - Cutting a Rod

3.1 Enunț

Dată fiind o bară de lungime n inch și un vector de prețuri care include prețurile tuturor bucăților de dimensiuni mai mici decât n, determinați valoarea maximă obținută prin tăierea barei și vânzarea bucăților obținute.

3.2 Date de intrare

* Datele se vor citi din fișierul „cr.in”
* O bară de lungime n, reprezentând lungimea inițială a barei de tăiat.
* Un vector de valori v, unde v[i] reprezintă valoarea asociată bucății obținute prin tăierea barei la lungimea i ($1 \leq i \leq n$).

3.3 Date de ieșire

Programul va afișa în fișierul „ cr.out” valoarea maximă obținută.

3.4 Input

8
1 8 6 9 10 17 20 20

3.5 Output

32

3.6 Pseudocod

```
cutRod(price, n)

        val[n+1]
        for i = 1 pana la n
            max_val = INT_MIN;
            for j = 0 pana la i - 1
                max_val <- max(max_val, price[j] + val[i-j-1]);
            end for
            val[i] <- max_val;
        end for

    intoarce val[n]

end cutRod
```

3.7 Descrierea soluției

Algoritmul construiește o tabelă val de dimensiune n+1 pentru a stoca valorile maxime de profit obținute pentru lungimi de batonuri de la 1 la n.

Descrierea pseudocodului:

1. Se inițializează val[0] cu 0, deoarece nu avem nicio lungime de bară disponibilă și, implicit, niciun profit.
2. Se utilizează două bucle for pentru a itera prin toate lungimile posibile de la 1 la n. Prima buclă (i) reprezintă lungimea barei, iar a doua buclă (j) reprezintă toate posibilele tăieri ale acesteia, de lungime i.
3. Pentru fiecare lungime i, se inițializează valoarea maximă cu o valoare minimă posibilă pentru a urmări maximul de profit posibil pentru acea lungime.
4. Se iterează prin toate tăierile posibile ale barei de lungime i folosind bucla j, și se actualizează valoarea maximă cu suma dintre prețul tăierii curente (price[j]) și valoarea maximă de profit obținută anterior pentru bara rămasă după tăiere (val[i-j-1]), unde i-j-1 reprezintă lungimea barei rămase după tăiere. Se actualizează val[i] cu valoarea maximă de profit obținută pentru lungimea de baton i.
5. După ce am iterat prin toate lungimile posibile, val[n] va conține valoarea maximă de profit obținută pentru bara de lungime n, care este răspunsul final al problemei.

3.8 Rularea pas cu pas a algoritmului pe inputul dat:

1. Inițializăm valoarea lui val[0] cu 0.
   val[0] = 0;
   n = 8;
   price = 1, 8, 6, 9, 10, 17, 20, 20;

2. Pentru fiecare lungime posibilă a tăieturii, calculăm cel mai mare preț pe care îl putem obține și stocăm rezultatul în val[i], începând cu i = 1 până la n.

```
Pentru i = 1 si j = 0, price[0] = 1, val[0] = 0, max_val = -2147483648
Pentru i = 2 si j = 0, price[0] = 1, val[1] = 1, max_val = -2147483648
Pentru i = 2 si j = 1, price[1] = 8, val[0] = 0, max_val = 2
Pentru i = 3 si j = 0, price[0] = 1, val[2] = 8, max_val = -2147483648
Pentru i = 3 si j = 1, price[1] = 8, val[1] = 1, max_val = 9
Pentru i = 3 si j = 2, price[2] = 6, val[0] = 0, max_val = 9
Pentru i = 4 si j = 0, price[0] = 1, val[3] = 9, max_val = -2147483648
Pentru i = 4 si j = 1, price[1] = 8, val[2] = 8, max_val = 10
Pentru i = 4 si j = 2, price[2] = 6, val[1] = 1, max_val = 16
Pentru i = 4 si j = 3, price[3] = 9, val[0] = 0, max_val = 16
Pentru i = 5 si j = 0, price[0] = 1, val[4] = 16, max_val = -2147483648
Pentru i = 5 si j = 1, price[1] = 8, val[3] = 9, max_val = 17
Pentru i = 5 si j = 2, price[2] = 6, val[2] = 8, max_val = 17
Pentru i = 5 si j = 3, price[3] = 9, val[1] = 1, max_val = 17
Pentru i = 5 si j = 4, price[4] = 10, val[0] = 0, max_val = 17
Pentru i = 6 si j = 0, price[0] = 1, val[5] = 17, max_val = -2147483648
Pentru i = 6 si j = 1, price[1] = 8, val[4] = 16, max_val = 18
Pentru i = 6 si j = 2, price[2] = 6, val[3] = 9, max_val = 24
Pentru i = 6 si j = 3, price[3] = 9, val[2] = 8, max_val = 24
Pentru i = 6 si j = 4, price[4] = 10, val[1] = 1, max_val = 24
Pentru i = 6 si j = 5, price[5] = 17, val[0] = 0, max_val = 24
Pentru i = 7 si j = 0, price[0] = 1, val[6] = 24, max_val = -2147483648
Pentru i = 7 si j = 1, price[1] = 8, val[5] = 17, max_val = 25
Pentru i = 7 si j = 2, price[2] = 6, val[4] = 16, max_val = 25
Pentru i = 7 si j = 3, price[3] = 9, val[3] = 9, max_val = 25
Pentru i = 7 si j = 4, price[4] = 10, val[2] = 8, max_val = 25
Pentru i = 7 si j = 5, price[5] = 17, val[1] = 1, max_val = 25
Pentru i = 7 si j = 6, price[6] = 20, val[0] = 0, max_val = 25
Pentru i = 8 si j = 0, price[0] = 1, val[7] = 25, max_val = -2147483648
Pentru i = 8 si j = 1, price[1] = 8, val[6] = 24, max_val = 26
Pentru i = 8 si j = 2, price[2] = 6, val[5] = 17, max_val = 32
Pentru i = 8 si j = 3, price[3] = 9, val[4] = 16, max_val = 32
Pentru i = 8 si j = 4, price[4] = 10, val[3] = 9, max_val = 32
Pentru i = 8 si j = 5, price[5] = 17, val[2] = 8, max_val = 32
Pentru i = 8 si j = 6, price[6] = 20, val[1] = 1, max_val = 32
Pentru i = 8 si j = 7, price[7] = 20, val[0] = 0, max_val = 32
```

3. Se returneaza val[n].

3.9 Complexitate

* **Complexitate temporală:** O(n*n)
- deoarece utilizăm un singur vector val[n+1] pentru a stoca valorile calculate. Nu folosim alte structuri de date auxiliare de dimensiuni variabile în funcție de intrare, deci spațiul de memorie utilizat este proporțional cu dimensiunea de intrare n.

* **Complexitate spațială:** O(n), deoarece utilizăm un singur vector val[n+1] pentru a stoca valorile calculate. Nu folosim alte structuri de date auxiliare de dimensiuni variabile în funcție de intrare, deci spațiul de memorie utilizat este proporțional cu dimensiunea de intrare n.

4 Backtracking - Fazan

4.1 Enunț

Fiind date numerele naturale n, si m și apoi n cuvinte distincte cu cel mult 10 litere mici fiecare, să se afișeze toate secventele de câte m cuvinte dintre cele citite care să respecte condițiile jocului "fazan".

4.2 Date de intrare

Se citesc din fisierul "fazan.in" numerele naturale n, și m, iar mai apoi, pe următoarea linie, n cuvinte distincte.

4.3 Date de ieșire

Se afișează in fișierul ”fazan.out” toate secvente de cate m cuvinte, care respectă regulile jocului ”fazan” , fiecare secventă fiind urmată de un newline.

4.4 Input

8 3
paul alina asfalt nas ultim imagine nasture real

4.5 Output

paul ultim imagine
alina nas asfalt
alina nasture real
nasture real alina
real alina nas
real alina nasture

4.6 Pseudocod

```
Funcția back(k)
    if k > m
        Afișează rezultatul
        return
    end if

    for i = 1 până la n
        if P[i] este fals
            X[k] = i
            P[i] = 1
            if k == 1 sau fazan(C[X[k - 1]], C[X[k]])
                Apel recursiv: back(k + 1)
            end if
            P[i] = 0
        end if
    end for
end Funcția back
```

4.7 Descrierea soluției

Funcția ”back” folosește tehnica recursivă de backtracking pentru a genera toate permutările. Aceasta primește un parametru k, care reprezintă poziția elementului în permutare. Inițial, k este 1. Funcția parcurge toate elementele din setul de n elemente și verifică dacă elementul respectiv a fost deja folosit în permutare, prin intermediul vectorului P, unde P[i] este 1 dacă elementul i a fost folosit și 0 altfel.

Dacă elementul nu a fost folosit, atunci acesta este adăugat în permutare la poziția k (X[k] = i), se marchează elementul ca fiind folosit (P[i] = 1) și se verifică dacă permutarea curentă este validă prin apelarea funcției "fazan" pentru ultimele două elemente adăugate în permutare și ultimele două elemente adăugate anterior în permutare.

Dacă permutarea este validă și k este egal cu m (numărul de elemente din permutare), atunci se afișează permutarea. În caz contrar, se continuă să se genereze permutări, prin apelarea recursivă a funcției "back" cu parametrul k+1, pentru a adăuga următorul element în permutare.

După terminarea funcției, toate permutările posibile au fost generate și afișate.

Funcția "fazan" primește două șiruri de caractere, x și y, și verifică dacă ultimele două caractere din x sunt aceleași cu primele două caractere din y. Dacă da, funcția returnează 1, altfel returnează 0.

4.8 Rularea pas cu pas a algoritmului pe inputul dat:

```
P[1]: P[2]: P[3]: P[4]: 1 0 0 0
X[4]: 1 5 6 0
fazan(C[X[4 - 1]], C[X[4]]): 0
k: 4
paul ultim imagine
P[2]: P[3]: P[4]: 0 1 1 1
X[4]: 2 4 3 0
fazan(C[X[4 - 1]], C[X[4]]): 0
k: 4
alina nas asfalt
P[3]: P[4]: 0 1 0 0
X[4]: 2 7 8 0
fazan(C[X[4 - 1]], C[X[4]]): 0
k: 4
alina nasture real
P[2]: P[2]: P[3]: P[2]: P[3]: P[2]: P[2]: P[3]: P[4]: 0 1 0 0
X[4]: 7 8 2 0
fazan(C[X[4 - 1]], C[X[4]]): 0
k: 4
nasture real alina
P[2]: P[3]: P[4]: 0 1 0 1
X[4]: 8 2 4 0
fazan(C[X[4 - 1]], C[X[4]]): 0
k: 4
real alina nas
P[4]: 0 1 0 0
X[4]: 8 2 7 0
fazan(C[X[4 - 1]], C[X[4]]): 0
k: 4
real alina nasture
```

4.9 Complexitate

* **Complexitate temporală:**O(n!)
* **Complexitate spațială:** O(n)

, unde n reprezintă numărul de cuvinte de intrare.

Bibliografie

1. "The Art of Computer Programming, Volume 1: Fundamental Algorithms" - Donald E. Knuth - Cap. Egyptian fractions (Section 1.2.8)
2. https://www.pbinfo.ro/probleme/82/minmax , accesat ultima dată in 19.04.2023
3. https:https://www.geeksforgeeks.org/greedy-algorithm-egyptian-fraction/ , accesat ultima dată in 19.04.2023
4. https://www.geeksforgeeks.org/cutting-a-rod-dp-13/ , accesat ultima dată in 19.04.2023
5. https://andrei.clubcisco.ro/2aa/diverse/Exercitii%20complexitate%20si%20recursivitate.pdf, accesat ultima dată 19.04.2023
6. https://info.mcip.ro/?cap=Backtracking&prob=818 , accesat ultima dată 20.04.2023
7. https://www.tutorialspoint.com/design_and_analysis_of_algorithms/design_and_analysis_of_algorithms_max_min_problem. , accesat ultima dată 20.04.2023