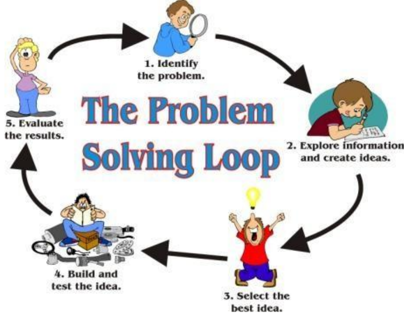
FUNDAMENTELE PROGRAMĂRII

Metode de rezolvare a problemelor

Metoda Greedy

Paşi în rezolvarea problemelor



- Definirea problemei
- Analiza problemei
- Alegerea unei tehnici de rezolvare

Metoda Greedy

- Ideea de bază
  - Descompunerea problemei în sub-probleme **succesive** și similare problemei inițiale, dar de dimensiuni mai mici, rezolvarea sub-problemelor și stabilirea soluției finale prin alegerea succesivă a celor mai bune sub-soluții
  - Optimul global = o succesiune de optime locale

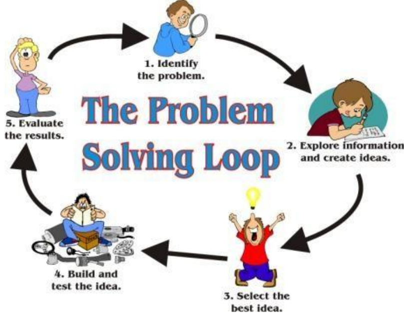
- Mecanism
  - împărțirea problemei în sub-probleme succesive P1, P2, ...Pn
  - Construirea treptată a soluției prin alegerea, la fiecare pas, a celei mai bune decizii

Greedy

- Când se poate folosi
  - Problema P (de optimizare)
  - Soluția este rezultatul unei succesiuni de optime locale
  - Pentru probleme a căror soluție este reprezentată prin submulțimi sau produse carteziene pentru care se atinge un anumit optim (minim sau maxim) al unei funcții obiectiv

- Caracteristici
  - Poate furniza soluția optimă
  - Construiește treptat soluția
  - Furnizează o singură soluție
  - Timp de lucru polinomial

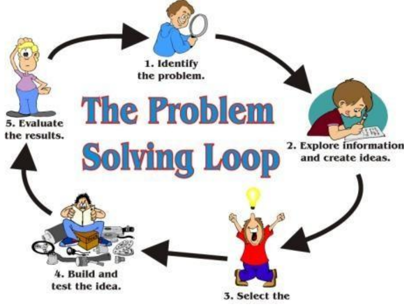
Paşi în rezolvarea problemelor



Preliminarii

- În procesul *proiectării algoritmilor de rezolvare a problemelor* o mare importanță o are alegerea **strategiei potrivite** (**optime**) de rezolvare.
- Strategia aleasă influențează asupra *duratei de rezolvare a problemei* cât și asupra *calității rezultatului obținut*.
- **Tehnica Greedy** ("*lacom*") este una dintre cele mai simple metode de elaborare a algoritmilor. Este folosită la rezolvarea **problemelor de optimizare**, **de minim**, respectiv **maxim**.
- Metoda Greedy reprezintă un *algoritm de tip "constructiv"* în sensul că, în rezolvarea problemei, se pornește de la o **soluție posibilă** (*inițială*), *soluție care apoi se îmbunătățește ajungându-se la rezultatul dorit pas cu pas.*

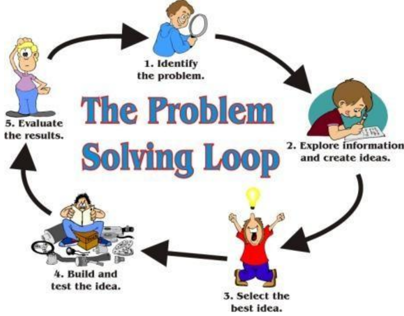
Paşi în rezolvarea problemelor



Mod de realizare

- La fiecare pas al unui algoritm greedy *se adaugă la soluție* un element optim sau promitator (se "*inghite*").
- Pentru implementare se consideră o mulțime finită $A = (a_1, a_2, ..., a_n)$, formată din n elemente, din care se aleg pe rând numai acele elemente care indeplinesc *anumite condiții, criterii* și se includ într-o submulțime $B \subseteq A$ (*mulțimea soluțiilor*).
- Alegând în orice moment *elementul optim pentru situația locală*, se asigură un *optim local*, dar nu se garantează că se va obține *optimul global*.
- Metoda dată determină întotdeauna o singură soluție a problemei.

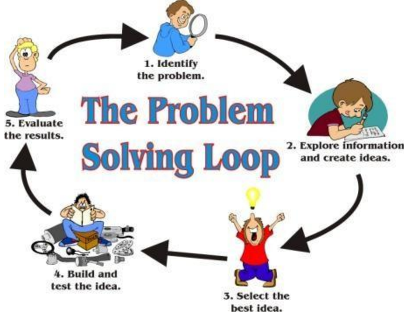
Paşi în rezolvarea problemelor



Observații

- În *metoda greedy*, spre deosebire de *metoda backtracking*, alegerea elementului x_k al soluției este **irevocabilă** (nu se mai poate reveni asupra alegerii făcute).
- Metoda greedy poate duce la obținerea *soluției optime* în cazul problemelor care au *proprietatea de optim local*, adică soluția optimă a problemei cu dimensiunea n a datelor de intrare *conține soluțiile optime ale subproblemelor* similare cu problema inițială, dar de dimensiune mai mică.
- Metoda greedy se mai numește și *metoda optimului local*.

Paşi în rezolvarea problemelor



Pașii algoritmului greedy

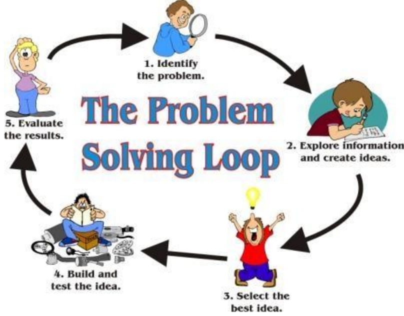
P1. Se inițializează submulțimea B cu *mulțimea vidă*: $B = \varnothing$.

P2. Cât timp B nu este soluție a problemei și $A \neq \varnothing$, execută:

- P3. Se alege din mulțimea A elementul a_i care este *candidatul optim al soluției*.
- P4. *Se elimină* elementul a_i din mulțimea A.
- P5. Dacă a_i poate fi *element al soluției*, atunci elementul a_i *se adaugă la mulțimea* B. Se revine la pasul P2.

P6. Dacă mulțimea B este *soluția problemei*, atunci se afișează soluția; altfel, se afișează mesajul "Nu s-a găsit soluție".

Paşi în rezolvarea problemelor

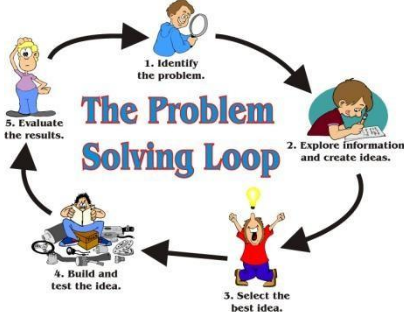


Condiții pentru obținerea soluției optime

1. Alegerea optimului local pentru fiecare element al soluției duce la alegerea soluției optime globale.
2. Soluția optimă a problemei conține soluțiile optime ale subproblemelor.

- Pentru a fi siguri că algoritmul greedy construiește soluția optimă a problemei, trebuie să se demonstreze că sunt îndeplinite cele două condiții.
- În rezolvarea multor probleme folosind strategia greedy, pentru a alege optimul local care să ducă la alegerea soluției optime globale, mulțimea A este *ordonată după criteriul candidatului optim*.
- Ordonarea mulțimii după criteriul candidatului optim înseamnă *rearanjarea elementelor mulțimii* A astfel încât, după extragerea unui element, *următorul element din mulțimea* A să reprezinte *elementul care este cel mai îndreptățit să fie ales ca element al mulțimii* B (*elementul cel mai promițător*).

Paşi în rezolvarea problemelor



Mecanismul metodei Greedy este următorul:

- Se pornește cu soluția vidă $\varnothing$.
- Se alege într-un anumit fel un element din A, neales la pașii precedenți.
- Se testează dacă elementul ales poate fi adăugat mulțimii B, inițial vidă.
- Dacă această adăugare la soluția parțial construită conduce la o soluție posibilă, atunci construim noua soluție posibilă prin adăugarea elementului.
- Procedeul continuă repetitiv în acest mod până când au fost alese toate elementele din A, care satisfac anumite condiții.

Greedy

Algoritm

- Fie *S* o soluție a problemei, iar *C* mulțimea optimelor locale pentru fiecare sub-problemă (mulțime de elemente candidate la soluție)

```
def greedy(C):
    S= Φ
    while (not Solutie(S)) and (C!=Φ):
        element = Selectare_Varianta_Optimă(C)
        C.remove(element)
        if Solutie_Acceptata(element, S):
            S.append(element)

    if Solutie(S)
        return S
    else:
        return None
```

Greedy

Exemplul:

- Să se găsească o modalitate de a plăti o sumă de bani folosind cât mai puține monezi de diferite valori.
  - Date: Suma = 80, Monezi = [1, 5, 10, 25, 50]
  - Rezultate: 80 = 50 + 25 + 5

  - Date: Suma = 10, Monezi = [1, 2, 3, 4]
  - Rezultate: 10 = 4 + 3 + 2 + 1

  - Date: Suma = 10, Monezi = [2, 3, 4, 5]
  - Rezultate: 10 = 5 + 3 + 2

Greedy

- SOLUȚIE
  - Să se găsească o modalitate de a plăti o sumă de bani folosind cât mai puține monezi de diferite valori.

```python
valoarea_monedei = [1, 5, 10, 25, 50]

def returnschimb(schimb, valoarea_monedei):
    returneaza = [0] * len(valoarea_monedei)

    for pos, moneda in enumerate(reversed(valoarea_monedei)):
        while moneda <= schimb:
            schimb = schimb - moneda
            returneaza[pos] += 1
    return(returneaza)

suma=int(input('Dati suma necesara='))
print('Valoarea monedei',[50,25,10,5,1])
print('Numarul de monede',returnschimb(suma, valoarea_monedei))
```

```
Dati suma necesara=54
Valoarea monedei [50, 25, 10, 5, 1]
Numarul de monede [1, 0, 0, 0, 4]
```


```python
valoarea_monedei = [1, 5, 10, 25, 50]

def returnschimb(schimb, valoarea_monedei):
    returneaza = [0] * len(valoarea_monedei)

    for pos, moneda in enumerate(reversed(valoarea_monedei)):
        while moneda <= schimb:
            schimb = schimb - moneda
            returneaza[pos] += 1
    return(returneaza)

suma=int(input('Dati suma necesara='))
print('Valoarea monedei',[50,25,10,5,1])
print('Numarul de monede',returnschimb(suma,
valoarea_monedei))
```

Materiale utile

1. Limbajul Python
   http://docs.python.org/3/reference/index.html

2. Biblioteca standard Python
   http://docs.python.org/3/library/index.html

3. Tutorial Python
   http://docs.python.org/3/tutorial/index.html

4. Martin Fowler. Refactoring. Improving the Design of Existing Code. *Addison-Wesley*, 1999 http://refactoring.com/catalog/index.html