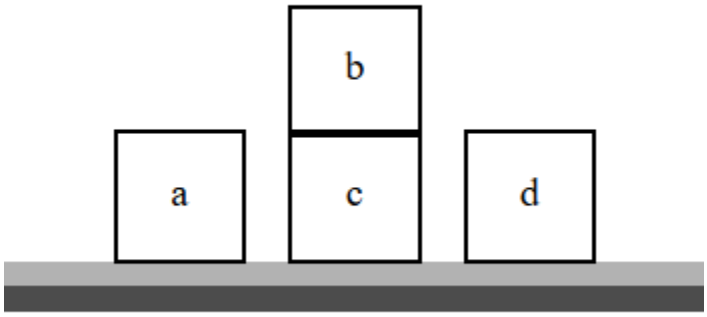
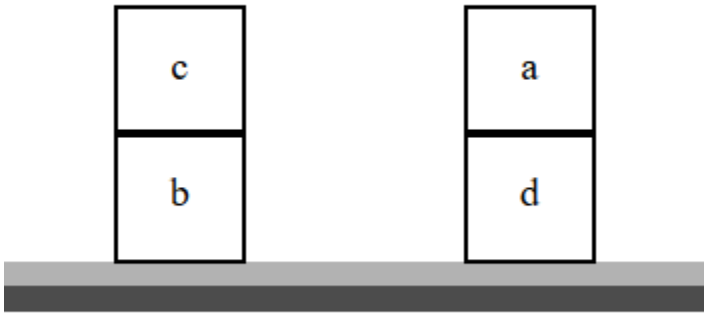
Problema blocurilor

Avem un număr N de blocuri(cuburi) dispuse pe S stive . Se dă o stare initială, cu blocurile așezate, într-o anumită configurație, de exemplu:



și una sau mai multe stări finale, cu blocurile rearanjate, de exemplu:



Dorim să obținem mutările (lista ordonată de stări intermediare) prin care ajungem de la starea inițială la o stare finală.

Exerciții

1) Folosiți clasele Graf și Nod implementate într-o cerință anterioară pentru algoritmul A*. Realizați următoarele cerințe pentru problema blocurilor.

a) Se vor citi dintr-un fișier starea inițială și stările finale. Atât starea inițială cât și cele finale vor fi reprezentate astfel: fiecare stivă din stare va fi pe câte un rând din fisier, blocurile de pe stivă sunt reprezentate prin informațiile lor (aici, o literă mică pe post de nume) separate prin spații. Stivele vide sunt reprezentate prin #. Starea inițială e separată fața de starile finale prin șirul "=========". Stările finale sunt separate cu "---". O stare va fi memorată ca o listă de stive. Stivele vor fi liste de blocuri. Pentru fiecare bloc se memorează datele care îl descriu (aici, numele).
b) Scrieți o metodă valideaza() în clasa Graf care verifică dacă fișierul dat e valid (toate stările finale au același număr de stive și aceleași blocuri ca și starea inițială).
c) Modificați funcția scop() astfel încât să verifice că informația nodului curent este printre informațiile scop date în fișier.

Exemplu de fișier de input:

```
a
c b
d
=========
b c
#
d a
---
a b c d
#
#
---
#
#
d b c a
---
c b d a
#
#
```

2) Modificați funcția succesori(nod) astfel încât să genereze toți succesorii valizi (a căror informație nu se repetă în istoricul lor), pentru această problemă (trebuie rezolvat pentru un număr de stive și de blocuri general; conform datelor citite din fișier, nu neapărat pentru 3 stive și 4 blocuri.). Creați două obiecte de tip Nod cu informații diferite și afișati succesorii pentru ele, verificând corectitudinea informațiilor.
Rulați cu BF și DF problema și verificați soluțiile din output.
Pentru rularea cu A* cerută mai jos se consideră costul pe o mutare egal cu 1.

3) Rezolvați problema blocurilor folosind A* urmând cerințele:

- Considerați costul pe orice mutare ca fiind numărul de ordine în alfabet

- Implementați funcția estimeaza_h() corespunzătoare problemei, funcția trebuind să returneze o estimație admisibilă pentru fiecare nod. Funcția va avea și un parametru numit euristică prin care se va alege tipul de euristică returnat. Funcția trebuie să admită 4 valori pentru euristică (primele 3 sunt admisibile):
    a. "banala" - caz în care va returna costul minim pe o mutare daca starea nu e scop și 0 dacă e scop
    b. "euristica mutari" - caz în care va returna un număr de mutări (mai mic sau egal decât numărul real de mutări astfel încât să ajungem de la starea curentă la cea mai apropiată stare scop) și 0 dacă starea curentă e scop. Un astfel de număr de mutări e dat de nb= "câte blocuri nu sunt la locul lor față de fiecare stare scop" și apoi luând minimul dintre aceste valori *nb*.
    c. "euristica costuri" - caz în care va returna un cost mai mic sau egal decât costul real al oricărui drum de la nodul curent la orice nod scop, folosindu-se în formulă și de costul de mutare al unui bloc. Veți verifica care sunt blocurile care nu sunt la locul lor și veți calcula suma costurilor lor de mutare. Pentru fiecare scop se va calcula o astfel de sumă din care returnăm minimul.
    d. "euristica neadmisibila" - caz in care se vor returna valori astfel încât estimația pentru nod să fie neadmisibilă.

  Exemplu de apelare: graf.estimeaza_h(nod, euristica="banala")

4) Creați o metodă, afisSolFisier() în clasa Nod, care afișează într-un fișier dat ca parametru (parametrul va fi o variabilă de tip fișier și nu o cale) soluțiile în formatul în care a fost afișat drumul de mai jos. Pentru afișați la fiecare pas costul de la start până la nodul curent afișat (costul parțial). La final afișați și costul total împreună cu lungimea drumului.

```
Solutie:
  b
a c d
-----
Cost partial: 0

    b
a c d
-----
Cost partial: 2

    c
    b
a   d
-----
Cost partial: 5
```

```
    a
    c
    b
    d
-----
Cost partial: 6

Cost:  6
Lungime:  4
```

5) Să se modifice problema blocurilor cu următoarele cerințe:

- nu putem pune un bloc pe o stivă dacă mai există în stivă un bloc cu aceeași informație.
- costul unei mutări este dat de înălțimea la care mutăm blocul (înăltimea unui bloc la baza stivei se consideră 1 și crește cu fiecare nivel).
- să consideram ca o stare este scop dacă toate stivele din ea sunt de înălțimi egale (indiferent ce avem pe stive). ATENTIE! nu veti enumera toate starile scop ci veti verifica daca o stare curenta indeplineste conditia data (nu mai avem lista de scopuri).

Se va implemeta și o funcție care verifică dacă din starea inițială e posibil să ajungem la o stare finală. Modificați funcția de succesori, de testare a scopului și de calculare a estimarii în mod corespunzător pentru noul criteriu de stare finală.