

KR LAB QUIZ seria 23 - LAB 1

1

Se consideră (în python) variabila:

`l=[2,7,10,3,5]`

Care dintre următoarele afirmații sunt adevărate? (15 puncte)

- nu există nicio funcție f astfel încât l.sort(key=f) sa schimbe l în [10, 1, 2, 23, 3, 5, 7, 77]
- Putem obține printr-o list comprehension de forma `[x for x in l if conditie(x)]` o lista cu mai putine elemente. Considerăm condiție(x) ca fiind o funcție cu rezultat boolean.
- Putem obține printr-o list comprehension de forma `[x for x in l if conditie(x)]` o lista cu mai multe elemente. Considerăm condiție(x) ca fiind o funcție cu rezultat boolean.
- Fiind data secevența python prin care cream o matrice (lista de liste) de dimensiune $n \times n$ (considerăm $n > 2$): `l=[[0 for _ in range(n)] for _ in range(n)]; l[0][0]=10` Va exista în matrice un singur element cu valoarea 10 și $n^2 - 1$ elemente nule.
- Fiind data secevența python prin care cream o matrice (lista de liste) de dimensiune $n \times n$ (considerăm $n > 2$): `a=[[0]*n]*n; a[0][0]=10; print(a);` Vor exista în matrice n elemente cu valoarea 10.

2

Considerăm codul din imagine prin care definim o clasă python.

Care dintre următoarele afirmații sunt adevărate: (15 puncte)

```
1 class Student:
2     NR=0
3     studentiCreati=[]
4
5     def __init__(self, _nume, _prenume, _grupa, _note=[]):
6         self._nume=_nume
7         self._prenume=_prenume
8         self._grupa=_grupa
9         self._note=_note
10        self.__class__._NR+=1
11        self.__class__._studentiCreati.append(self)
12
13    def calculeaza_mediile(self):
14        return self._note/len(self._note)
15
16    def __str__(self):
17        return "Studentul "+self._nume+" "+self._prenume
18
19
20    def __repr__(self):
21        return "({},{},{})".format(self._nume, self._prenume, self._grupa)
22
23    @classmethod
24    def resetazaStudenti(cls):
25        cls.NR=0
26
27 s1=Student("Ionescu", "Ionel", 213, [10,9,5])
28 s2=Student("Costescu", "Costel", 223, [10,9,10,10])
```

- `print(s1)` va afisa: Studentul Ionescu Ionel
- `print(s1)` va afisa: (Ionescu, Ionel, 213)
- funcția `__init__` este constructorul clasei și trebuie apelată întotdeauna cu exact 4 argumente
- Dacă scriem în program, fără alte modificări `s1.NR=100; print(Student.NR)` va afișa 100
- Dacă scriem `print(s1 < s2)` vom primi eroare și putem rezolva acest lucru definind metoda `__lt__` în cadrul clasei

3

Care dintre următoare afirmații sunt adevărate: (15 puncte)

- O problemă ale căreia și tranziții între stări pot fi abstractizate la nodurile unui arbore și legăturile dintre acestea, poate fi rezolvată cu ajutorul unei tehnici de căutare **doar dacă** rădăcina este nodul start și nodurile scop sunt frunze.
- Dacă nodul start și nodurile scop nu se află pe aceeași componentă conexă, problema nu are soluție.
- Dacă aplicăm atât BreadthFirst cât și DepthFirst pe o problemă oarecare, prima soluție oferită de BF nu va coincide **niciodată** cu prima soluție oferită de DF.
- Soluțiile oferite de algoritmul BreadthFirst (presupunând că îl rulăm pentru a obține toate drumurile posibile de la nodul start la toate nodurile scop) sun returnate în ordinea lungimii lor.
- Algoritmul DepthFirst folosește o structură de tip coadă pentru a-și construi drumurile.
- Dacă nodul start este nod izolat în graf, mulțimea soluțiilor este 0 indiferent care este mulțimea nodurilor scop din graf.

4

Considerăm un graf orientat cu $n > 2$ noduri numerotate de la 1 la n (numărul va fi informația nodului). Vom nota informația unui nod cu `info(nod)`. Considerând că mulțimea arcelor A este mulțimea tuturor perechilor de noduri (i,j) (arcul este de la i la j) cu $info(i) \neq info(j)$ și are proprietatea că $info(i)$ este divizor al lui $info(j)$. Vom considera scopurile ca fiind nodurile cu informația divizibilă cu k, o constantă. Nodul start va fi considerat cel cu informația 1. Considerăm lungimea unui drum ca fiind numărul de muchii din drum. Care dintre următoarele afirmații sunt adevărate? (15 puncte)

- Pentru $k=5$ și orice n , algoritmul BreadthFirst va returna ca primă soluție un drum de lungime 5.
- Pentru $k=5$ și orice n , algoritmul BreadthFirst va returna ca primă soluție un drum de lungime 1.
- Pentru $k=5$ și orice $n > 5$, algoritmul BreadthFirst va returna ca primă soluție un drum de lungime 1.
- Pentru $k=5$ și orice $n > 5$, algoritmul BreadthFirst va returna ca primă soluție un drum de lungime 5.
- Pentru $k=5$ și orice $n > 10$, algoritmul DepthFirst, presupunând că succesorii sunt evaluati în ordinea crescătoare valorilor din nod, va returna ca primă soluție un drum de lungime 3.

LAB 2

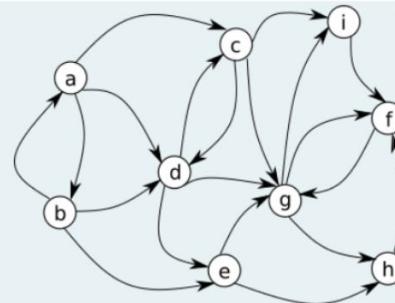
1

Considerăm graful din imaginea alăturată.

Vom considera că ordinea în care sunt evaluati succesorii pentru fiecare nod este ordinea alfabetica a informatiilor.

Considerăm lungimea unui drum ca fiind numărul de muchii din drum.

Care dintre următoarele afirmații sunt adevărate? (15 puncte)



- Nu există niciun nod start astfel încât, în cazul algoritmului BreadthFirst, coada de noduri să cuprindă noduri cu informațiile: a,b,c,d (scrise aici în ordinea nodurilor din coadă).
- Dacă nodul *a* e nod start și toate nodurile conținând o vocală sunt noduri scop, atunci lungimea, primului drum returnat de BreadthFirst e 2.
- Dacă nodul *a* e nod start și toate nodurile conținând o vocală sunt noduri scop, atunci lungimea, primului drum returnat de BreadthFirst e 0.
- În cazul în care nodul start este *a* și nodurile scop sunt *b* și *d*, primul drum soluție calculat de BreadthFirst e egal cu primul drum soluție calculat de DepthFirst
- În cazul în care nodul start este *a* și nodurile scop sunt *e* și *h*, primul drum soluție calculat de BreadthFirst e egal cu primul drum soluție calculat de DepthFirst

3

Care dintre următoarele fraze sunt adevărate: (15 puncte)

- Presupunem că avem o problemă care poate fi abstractizată la un graf. Aplicarea unei tehnici de căutare asupra problemei în scopul găsirii tuturor drumurilor soluție este echivalentă cu parcurgerea grafului (prin vizitarea și evaluarea o singură dată a tuturor nodurilor grafului, pe tot parcursul algoritmului)
- Problema creării orarului pentru profesorii dintr-o școală poate fi rezolvată folosind o tehnică de căutare considerând nodul inițial ca fiind tabelul de ore fără nicio lecție setată, iar reprezentarea unei stări ca fiind orele deja planificate până în momentul t al stării. O tranziție ar fi plasarea unei noi lecții în tabel.
- Se poate încerca problema vindecării unui bolnav cu diagnostic necunoscut folosind o tehnică de căutare, considerând nodul inițial ca fiind pacientul care nu a primit medicamentele deloc, o stare ca fiind pacientul căruia i s-au dat anumite medicamente până în momentul t și tranziția ca fiind oferirea unui medicament nou pacientului (din multimea tuturor medicamentelor disponibile în spital) pentru a vedea dacă îl vindecă. Vom presupune că medicii care îl tratează nu sunt oameni malefici și chiar le pasă de pacient.
- Se poate încerca problema vindecării unui bolnav cu diagnostic necunoscut folosind un sistem expert cu reguli pe baza cunoștințelor medicale, în scopul obținerii celui mai probabil diagnostic și tratamentului corespunzător.
- Funcția succesor aplicată unei stări x returnează toate nodurile din subarborele avându-l ca rădăcină pe x, din cadrul arborelui reprezentând spațiul de căutare.
- O problemă de căutare poate să aibă oricăte noduri scop.

2

Considerăm următoarea problemă de căutare:

Se citesc dintr-un fișier numerele P, G, B, B1.

Un țăran vrea să mute de pe malul stâng al unui râu pe malul drept P pisici, G gaini și B saci cu boabe.

Țăranul are la dispoziție o barcă aflată la început pe malul stâng. Țăranul e singurul care poate muta barca (nu se pot deplasa pisici, gaini sau saci cu boabe fără țăran).

Atunci când barca pleacă de pe un mal (deci pe acel mal nu mai este țăranul), dacă există și pisici și saci (intacti) cu boabe, acestea vor alege un sac cu boabe și-l vor zgropăna până se deșira, transformându-se în sac cu boabe desfăcut (chiar dacă sunt mai multe pisici, vor alege un singur sac cu boabe). De asemenea, pisicile nu au voie să rămână mai multe decât dublul numărului de gaini pe mal, fără țăran, fiindcă le vor ataca. Scopul țăranului este să mute, P pisici G gaini și B1 < B saci intacti cu boabe pe malul drept.

Bifați răspunsurile cu contextele (variabilele globale sau proprietățile statice pe care le avem) și structurile care pot reprezenta starea problemei **cu tot necesarul de informații pentru a cunoaște complet starea și a genera succesiuni pentru ea, dar în același timp, starea să nu conțină informații redundante**. Considerăm elementele din stare ca fiind numerele la momentul de timp în care se găsește starea. (15 puncte)

Salvăm numere P,G în variabile globale pe care nu le mai schimbăm. Starea poate fi reprezentată, în mod eficient, printr-un tuplu (sau o listă) cu valorile (numar_pisici_mal_stang, numar_gaini_mal_stang,

- numar_saci_intacti_mal_stang, numar_tarani_mal_stang, total_saci_intacti), unde total_saci_intacti se referă la suma numărului de saci intacti de pe ambele maluri.

Salvăm numere P,G,B în variabile globale pe care nu le mai schimbăm. Starea poate fi reprezentată, în mod eficient, printr-un tuplu (sau o listă) cu valorile (numar_pisici_mal_stang, numar_gaini_mal_stang, numar_saci_intacti_mal_stang, numar_tarani_mal_stang)

- Salvăm numere P,G,B în variabile globale pe care nu le mai schimbăm. Starea poate fi reprezentată, în mod eficient, printr-un tuplu (sau o listă) cu valorile (numar_pisici_mal_stang, numar_gaini_mal_stang, numar_saci_intacti_mal_stang, numar_saci_desirati_mal_stang, locatie_barca)

Salvăm numere P,G,B în variabile globale pe care nu le mai schimbăm. Starea poate fi reprezentată, în mod eficient, printr-un tuplu (sau o listă) cu valorile (numar_pisici_mal_drept, numar_gaini_mal_drept, numar_saci_intacti_mal_stang, numar_saci_desirati_mal_drept, locatie_barca)

- Salvăm numere P,G,B în variabile globale pe care nu le mai schimbăm. Starea poate fi reprezentată, în mod eficient, printr-un tuplu (sau o listă) cu valorile (numar_pisici_mal_stang, numar_pisici_mal_drept, numar_gaini_mal_stang, numar_gaini_mal_drept, numar_saci_intacti_mal_stang, numar_saci_intacti_mal_drept, locatie_barca)

Salvăm numere P,G,B în variabile globale pe care nu le mai schimbăm. Starea poate fi reprezentată, în mod eficient, printr-un tuplu (sau o listă) cu valorile (numar_pisici_mal_stang, numar_pisici_mal_drept, numar_gaini_mal_stang, numar_gaini_mal_drept, numar_saci_intacti_mal_stang, numar_saci_intacti_mal_drept, locatie_barca, total_saci_intacti)

Consideram problema canibalilor și misionarilor.

Pe malul unui râu se află N misionari, N canibali și o barcă cu M locuri (N și M numere naturale). Scopul e ca toți oamenii să treacă râul cu barca. În nicio locație nu avem voie să avenim mai mulți canibali decât misionari deoarece, în acest caz, canibali îi vor ataca pe misionari (stare invalidă). Barca nu poate pleca vînd.

Considerăm reprezentarea stării sub forma unui tuplu (numar_misionari_mal_initial, numar_canibali_mal_initial, mal_current), unde *mal_current* reprezintă malul pe care se află barca. Vom nota malul inițial cu 1 și final cu 0.

Considerăm o tranziție ca fiind o singură deplasare cu barca (de la un mal la altul). Întoarcerea bărcii ar fi o nouă tranziție.

Considerăm lungimea unei soluții (drum de stări de la nodul start până la scop) ca fiind numărul de muchii din drum.

Care dintre următoarele afirmații sunt adevărate? (15 puncte)

- Pentru $M \geq 4$, indiferent care ar fi $N \geq 1$, problema are întotdeauna soluție.
- Starea inițială a problemei pentru orice M și N are un număr de succesiuni egal cu $M^*(M+1)/2$.
- Pentru M și N egale, mai mari strict decât 1, cea mai scurtă soluție are lungime 5.
- Pentru M și N egale, mai mari strict decât 1, cea mai scurtă soluție are lungime 3.
- Pentru M și N egale, mai mari strict decât 1, nu se poate spune care este lungimea celei mai scurte soluții.
- Presupunând că $N=3$ și $M=2$, un succesor al stării inițiale este (2,2,0).
- Presupunând că $N=3$ și $M=2$, un succesor al stării inițiale este (3,2,1).

LAB 3

Observație: ponderile de pe toate arcele și muchiile grafurilor din probleme sunt numere strict pozitive.

1. Care dintre următoarele fraze sunt adevărate? (15 puncte)

- Pentru o evaluare euristică admisibilă, algoritmul A* oferă ca soluție drumul de cost minim, indiferent de numărul de muchii din drumul returnat.
- Pentru o evaluare euristică admisibilă, algoritmul A* oferă ca primă soluție drumul de lungime minimă (număr minim de muchii în drum), indiferent de costul muchiilor.
- Frontiera, în cadrul tehnicilor de căutare, reprezintă colecția de noduri descoperite din graf care urmează să fie extinse.
- Frontiera, în cadrul tehnicilor de căutare, reprezintă colecția de noduri descoperite din graf care au fost extinse.

2. Care dintre următoarele fraze sunt adevărate? (15 puncte)

- Se consideră un factor euristic admisibil. Algoritmul A* poate fi folosit pentru calcularea drumului de lungime maximă prin înmulțirea costului de pe fiecare arc cu -1, dar și înmulțirea cu -1 a factorului euristic $h(n)$. În acest caz se ia primul drum returnat de algoritmul modificat, considerând costul său ca fiind cel calculat de A*-modificat înmulțit cu -1.
- Toate stările corespunzătoare nodurilor dintr-un drum soluție returnat de A* sunt distincte.
 - Algoritmul A*, în cazul oricărei probleme, construiește arborele de căutare nivel de nivel, generând întâi toate nodurile de pe nivelul 1, urmate imediat de toate nodurile de pe nivelul $k+1$, pentru orice k natural, până când adaugă în arbore un nod scop.
 - Algoritmul A* alege întotdeauna pentru expandare unul dintre nodurile n (din lista OPEN) care au valoarea $f(n)$ minimă.

4. Care dintre următoarele fraze sunt adevărate?

Observație (necesară pentru unele subpunkte): Vom presupune că nodurile de cost egal sunt evaluate mereu după aceeași relație de ordine pe stări. De exemplu, dacă relația de ordine pe stări este cea alfabetică, dacă în coadă avem nodurile b și c ambele cu costul estimat \hat{h} egal cu 5, b va fi înainte de c. De asemenea, presupunem că estimările pentru graful de dinainte de eventuala modificare (din fiecare subpunkt) sunt admisibile. (15 puncte)

- Considerăm un graf neorientat oarecare cu un nod start și un singur nod scop z, iar soluția returnată de A* este (a, n_1, n_2, ..., n_k, z), unde n_1, n_2, ..., n_k sunt noduri din graf. Dacă schimbăm problema, considerând nodul start, ca fiind z, și unicul nod scop ca fiind a, și modificând estimările \hat{h} astfel încât să fie în continuare admisibile, atunci drumul returnat de A* este întotdeauna (z, n_k, ..., n_2, n_1, a).

- Considerăm un graf neorientat oarecare cu un nod start și un singur nod scop z, iar soluția returnată de A* este (a, n_1, n_2, ..., n_k, z), unde n_1, n_2, ..., n_k sunt noduri din graf. Dacă schimbăm problema, considerând nodul start ca fiind z, și unicul nod scop ca fiind n_i, unde n_i este unul dintre nodurile n_1, ..., n_k, și modificând estimările \hat{h} astfel încât să fie în continuare admisibile, atunci drumul returnat de A* este întotdeauna (z, n_k, ..., n_i).

- Considerăm un graf neorientat cu un nod start și cu mai multe noduri scop dintre care unul este z, iar soluția returnată de A* este (a, n_1, n_2, ..., n_k, z), unde n_1, n_2, ..., n_k sunt noduri din graf. În acest caz putem spune cu certitudine că niciunul dintre nodurile n_1, n_2, ..., n_k nu este nod scop.

- Considerăm un graf neorientat oarecare cu un nod start și un singur nod scop z, iar soluția returnată de A* este d=(a, n_1, n_2, ..., n_k, z), unde n_1, n_2, ..., n_k sunt noduri din graf. Dacă schimbăm problema, considerând nodul start, ca fiind z, și unicul nod scop ca fiind a, și modificând estimările \hat{h} astfel încât să fie în continuare admisibile, atunci drumul returnat de A* de la z la a, e întotdeauna de aceeași lungime cu d.

- Considerăm un graf neorientat oarecare cu un nod start și un singur nod scop z, iar soluția returnată de A* este d=(a, n_1, n_2, ..., n_k, z), unde n_1, n_2, ..., n_k sunt noduri din graf. Dacă schimbăm problema, considerând nodul start, ca fiind z, și unicul nod scop ca fiind a, dar lăsăm valorile estimărilor \hat{h} , atunci drumul returnat de A* de la z la a, e întotdeauna de aceeași lungime cu d.

3. Care dintre următoarele fraze sunt adevărate? (15 puncte)

- A* e o tehnică de căutare neinformată.
- A* e o tehnică de căutare informată.
- În cazul oricărui graf, algoritmul A* întotdeauna evaluează (expandează) toate nodurile existente în graf pe parcursul căutării pentru a găsi o soluție.
- Dacă există lanțuri/drumuri de la nodul start la unul sau mai multe noduri scop atunci A* sigur va returna o soluție.

5. Considerăm că graful problemei este un graf oarecare, neorientat, de tip arbore având muchii cu ponderi numere naturale mai mari decât 1. (15 puncte)

- Dacă și nodul start și nodurile scop sunt frunze (diferite de nodul start), atunci, întotdeauna, nodul rădăcină din graful problemei va fi evaluat și expandat exact o singură dată de algoritmul A* până la oferirea soluției.
- Dacă și nodul start și nodurile scop sunt frunze (diferite de nodul start), atunci atât părintele nodului start, cât și părintele nodului scop (din soluție) vor fi întotdeauna evaluate și expandate exact o singură dată de algoritmul A* până la oferirea soluției.
- Dacă și nodul start și nodurile scop sunt frunze (diferite de nodul start), atunci fiecare nod existent în graful problemei va fi evaluat și expandat exact o singură dată de algoritmul A* până la oferirea soluției.
- Dacă nodul start este rădăcina și nodurile scop sunt frunze, drumul returnat de A* întotdeauna se va termina cu nodul scop aflat la cea mai mică adâncime.

6. Considerăm că graful problemei este un graf neorientat de tip arbore cu muchii ponderate și costuri numere naturale mai mari sau egale cu 1.

Vom presupune că nodurile de cost egal sunt evaluate mereu după aceeași relație de ordine pe stări atât în A* cât și în BreadthFirst.

Presupunem că lucrăm cu un factor heuristic admisibil.

Adâncimea unui nod n este numărul de muchii aflate în lanțul de la rădăcină la nodul n .

Bifați afirmațiile adevărate: (15 puncte)

Dacă și nodul start și nodurile scop sunt frunze (diferite de nodul start). Pentru fiecare legătură (muchie) $m=(n_1, n_2)$ din graf, alegem o funcție de cost

- $\text{cost}(m)=k*\min_a+1$, unde \min_a reprezintă minimul dintre adâncimile nodurilor și $k>2$. În această situație, primul drum soluție returnat de A* este mereu și primul drum soluție returnat de BreadthFirst.

Dacă și nodul start e rădăcina și nodurile scop sunt frunze. Pentru fiecare legătură (muchie) $m=(n_1, n_2)$ din graf, alegem o funcție de cost

- $\text{cost}(m)=k*\min_a+1$, unde \min_a reprezintă minimul dintre adâncimile nodurilor și $k>2$. În această situație, primul drum soluție returnat de A* este mereu și primul drum soluție returnat de BreadthFirst.

Pentru fiecare legătură (muchie) $m=(n_1, n_2)$ din graf, alegem o funcție de

- cost $\text{cost}(m)=1$, atunci indiferent care este nodul start și care sunt nodurile scop, primul drum soluție returnat de A* este mereu și primul drum soluție returnat de BreadthFirst.

Dacă și nodul start și nodurile scop sunt frunze (diferite de nodul start). Pentru fiecare legătură (muchie) $m=(n_1, n_2)$ din graf, alegem o funcție de cost

- cost $\text{cost}(m)=\text{ad_max} - \max_a + 1$, unde \max_a reprezintă maximul dintre adâncimile nodurilor n_1 și n_2 , iar ad_max este adâncimea maximă a grafului. În această situație, primul drum soluție returnat de A* este mereu și primul drum soluție returnat de BreadthFirst.

7. Care dintre următoarele fraze referitoare la notațiile și formulele folosite în A* sunt adevărate? Notăm cu n un nod oarecare din graf. (15 puncte)

- $\hat{g}(n)$ este factorul heuristic, $\hat{h}(n)$ este factorul de adâncime, iar funcția heuristică de evaluare are formula $\hat{f}(n)=\hat{g}(n)-\hat{h}(n)$
- $\hat{g}(n)$ este factorul de adâncime, $\hat{h}(n)$ este factorul heuristic, iar funcția heuristică de evaluare are formula $\hat{f}(n)=\hat{g}(n)+\hat{h}(n)$
- Condiția de admisibilitate pentru estimarea \hat{h} este $\hat{h}(n) \leq h(n)$
- În cazul în care estimarea \hat{h} este admisibilă avem că $\hat{f}(n)$ este mai mic sau egal decât costul oricărui drum de la nodul start la nodul scop, trecand prin n .

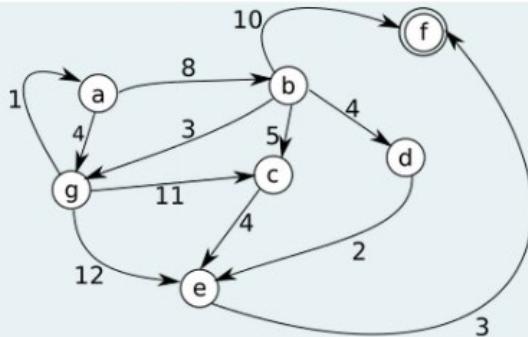
1

Pentru graful din imagine avem următoarele stări ale cozii OPEN (ordonate). Prima stare e cea inițială a cozii open (cu nodul start inclus). Toate celelalte stări sunt afișate în urma repetării realizării acțiunii compuse din:

- 1) extragerea nodului cu f^* minim (nu se afisează coada pentru această stare intermedie)
- 2) adăugarea succesorilor în coadă astfel încât coada să fie mereu ordonată crescător după f^* . (după această acțiune se afișează coada).

Observație: Pentru valori f^* egale, elementele sunt în ordinea descrescătoare a valorii g . În cazul în care și f și g sunt egale, nodurile se ordonează după informație.

Nodul start este a, iar nodul scop este f.



Stările cozii OPEN:

- [a]
- [b, g]
- [g, f, d, c]
- [c, f, d, e]
- [f, e, d]

Care dintre următoarele estimații (scrise între paranteze drepte pentru fiecare nod) ar fi în același timp admisibile și ar reprezenta și valori potrivite astfel încât coada OPEN a algoritmului A* să treacă prin valorile de mai sus?

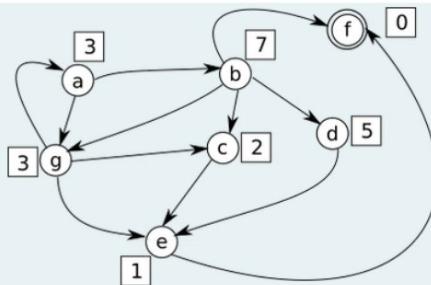
(15 puncte)

- a[10] b[1] c[3] d[5] e[4] f[0] g[6]
- a[1] b[2] c[5] d[1] e[4] f[1] g[6]
- a[0] b[0] c[0] d[0] e[0] f[0] g[0]
- a[1] b[1] c[1] d[1] e[1] f[1] g[1]
- nu există un set de estimații astfel încât coada să treacă prin valorile date

2

Pentru care dintre funcțiile de cost de mai jos (care asociază unui arc costul său), toate estimările, scrise în pătrătele, ale nodurilor din graful dat în imagine, sunt admisibile. Notăm cu n_1 și n_2 nodurile între care se găsește arcul $n_1 \rightarrow n_2$. Nodul start este a , nodul scop este f .

Am considerat:
 $\text{ord}(\text{nod})$ =codul ASCII al literei nodului,
 $\text{abs}(\text{numar})$ returnează valoarea absolută a numărului,
 $\text{grad_intern}(\text{nod})$ returnează gradul intern al nodului (câte arce intră în nod). **(15 puncte)**

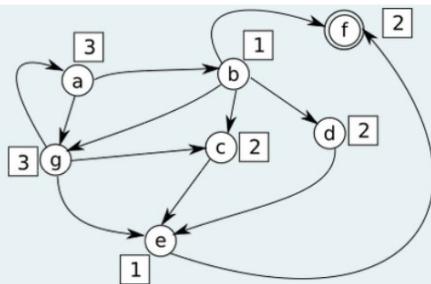


- $\text{cost}(n_1, n_2) = 1$, oricare ar fi n_1 și n_2
- $\text{cost}(n_1, n_2) = \text{abs}(\text{ord}(n_1) - \text{ord}(n_2)) * 10$, unde $\text{ord}(n)$ =codul ASCII al literei nodului, iar $\text{abs}(\text{numar})$ returnează valoarea absolută
- $\text{cost}(n_1, n_2) = 7$, oricare ar fi n_1 și n_2
- $\text{cost}(n_1, n_2) = \text{abs}(\text{ord}(n_1) - \text{ord}(n_2))$
- $\text{cost}(n_1, n_2) = \text{abs}(\text{grad_intern}(n_1) - \text{grad_intern}(n_2)) + 1$
- nu există o funcție de cost care să îndeplinească cerința problemei

3

Pentru care dintre funcțiile de cost de mai jos (care asociază unui arc costul său), toate estimările, scrise în pătrătele, ale nodurilor din graful dat în imagine, sunt admisibile. Notăm cu n_1 și n_2 nodurile între care se găsește muchia $n_1 \rightarrow n_2$. Nodul start este a , nodul scop este f .

Am considerat:
 $\text{ord}(\text{nod})$ =codul ASCII al literei nodului,
 $\text{abs}(\text{numar})$ returnează valoarea absolută a numărului,
 $\text{grad_intern}(\text{nod})$ returnează gradul intern al nodului (câte arce intră în nod), $\text{sqrt}(\text{numar}) = \text{radacina pătrată a numărului}$. **(15 puncte)**



- $\text{cost}(n_1, n_2) = 1$, oricare ar fi n_1 și n_2
- $\text{cost}(n_1, n_2) = \text{abs}(\text{ord}(n_1) - \text{ord}(n_2)) * 10000$, unde $\text{ord}(n)$ =codul ASCII al literei nodului, iar $\text{abs}(\text{numar})$ returnează valoarea absolută
- $\text{cost}(n_1, n_2) = 3$, oricare ar fi n_1 și n_2
- $\text{cost}(n_1, n_2) = \text{ord}(n_1) + \text{ord}(n_2)$
- $\text{cost}(n_1, n_2) = \text{abs}(\text{grad_intern}(n_1) - \text{grad_intern}(n_2)) + \text{sqrt}(\text{ord}(n_1) + 10 * \text{ord}(n_2))$
- nu există o funcție de cost care să îndeplinească cerința problemei

5

Pentru problema canibabliilor și misionarilor cu N canibali, N misionari și M locuri în barcă, considerăm starea curentă $st=(nmi, nci, b)$, unde nmi reprezintă numărul de misionari aflați acum pe malul initial, nci reprezintă numărul de canibali aflați curent pe malul initial iar b e locația bărcii ($b=1$ pentru barca pe malul initial și $b=0$ pentru barca pe malul final). Considerăm costul pe o mutare egal cu 1. Care dintre formulele pentru funcția de estimare duc la o funcție \hat{h} admisibilă: (15 puncte)

- $\hat{h}(st)=\max(nci, nmi)/M$
- $\hat{h}(st)=\min(N-nci, N-nmi)$
- $\hat{h}(st)=\max(0, 2*(nmi+nci)/M - 1)$
- $\hat{h}(st)=2*(nmi+nci)/(M - 1) - 1 *b$
- $\hat{h}(st)=2*(nmi+nci)/(M - 1)$
- $\hat{h}(st)=1$

4

Care dintre afirmațiile următoare sunt adevărate (notațiile sunt cele în prezentarea algoritmului A*): (15 puncte)

- În coada OPEN nu putem avea (în același timp) două noduri cu aceeași informație
- În coada OPEN nu vom găsi niciodată mai mult de un nod scop.
- În lista CLOSED nu vom găsi niciodată mai mult de un nod scop.
- Estimația nodului start nu poate fi niciodată 0.
- Un nod nu se poate găsi și în OPEN și în CLOSED în același timp.

LAB 5

Se consideră cunoscu enunțul clasic al problemei blocurilor:

Aveam un număr N de blocuri (cuburi) dispuse pe S stive. Se dă o stare initială, cu blocurile așezate, într-o anumită configurație, și una sau mai multe stări finale, cu blocurile rearanjate. Dorim să obținem mutările (lista ordonată de stări intermedii) prin care ajungem de la starea inițială la o stare finală.

Considerăm costul unei mutări egal cu 1.

1

Considerăm problema blocurilor cu următoarea modificare. Blocurile se pot muta doar pe o stivă imediat vecină (de pe stiva i putem muta un bloc doar pe $i+1$ sau $i-1$, dacă aceste stive există) (15 puncte)

- Orice estimație admisibilă pentru problema blocurilor clasice este admisibilă și pentru această problemă.
- Numărul de succesiuni pentru orice stare este S^2
- Numărul de succesiuni pentru orice stare este S^2-2
- Dacă notăm cu Sv numărul de stive vide în starea curentă, numărul de succesiuni ai stării curente este $(S-Sv)^2-2$
- Dacă notăm cu Sv numărul de stive vide în starea curentă, numărul de succesiuni ai stării curente este $(S-Sv)^2(S-1)$

3

Pentru o problemă oarecare (deci afirmația trebuie să fie adevărată pentru orice problemă de căutare cu costuri strict pozitive pe tranziții), care dintre formulele următoare ar obține din **estimația admisibilă \hat{h} (nod)**, mai mare sau egală cu 0, oricare ar fi aceasta (adică formula trebuie să fie adevărată pentru orice \hat{h} nu pentru cazuri particulare de \hat{h}), **o nouă estimație $\hat{h}_1(nod)$ în mod cert admisibilă?** Observație: în caz că nu se specifică nimic despre nodul dat ca parametru, înseamnă că se aplică formula pentru orice nod din graf, de orice tip. (15 puncte)

2

Considerăm problema blocurilor cu următoarea modificare. Nu putem să mutăm un bloc pe o stivă imediat vecină (nu pot muta de pe stiva i pe stiva $i+1$ sau $i-1$). Scopurile nu mai sunt date sub formă de configurații. Scopul este dat ca o condiție: și anume dorim să ajungem într-o configurație în care toate blocurile să fie adunate într-o singură stivă (oricare ar fi aceasta și oricare ar fi ordinea blocurilor.)

Care afirmații sunt adevărate? (15 puncte)

- Numărul stăriilor scop este $N!$
- Numărul stăriilor scop este $(N+1)!$
- Numărul stăriilor scop este $S^*N!$
- O estimație admisibilă pentru o stare cu Sv stive vide este $\hat{h}(\text{stare}) = \max(0, S - Sv - 1)$
- O estimație admisibilă pentru o stare este $\hat{h}(\text{stare}) = \text{"numărul tuturor blocurilor care nu se găsesc pe una din stivele de înălțime maximă"}$
- Pentru o stare inițială cu $S=3$ și nicio stivă vidă, nu există drum soluție.
- $\hat{h}_1(\text{nod}) = \hat{h}(\text{nod})/3$
- $\hat{h}_1(\text{nod}) = 1/\hat{h}(\text{nod})$
- $\hat{h}_1(\text{nod}) = \hat{h}(\text{nod}) * \hat{h}(\text{nod})$
- $\hat{h}_1(\text{nod}) = \min(\hat{h}(\text{nod}) + 2, \max(0, \hat{h}(\text{nod}) - 2))$
- Presupunând că mai avem încă o estimație admisibilă $\hat{h}_2(\text{nod})$, $\hat{h}_1(\text{nod}) = (\hat{h}(\text{nod}) + \hat{h}_2(\text{nod}))/2$

4

Se consideră următoarea problemă a blocurilor modificată.

Bocurile alergătoare.
(enunțul e identic cu cealaltă problemă numită "Blocuri alergătoare").

Avem S stive cu $S \geq 3$ și N blocuri, cu $N \geq 1$. În starea inițială, prima stivă (cea mai din stânga) conține toate blocurile.

Fiecare bloc are un număr (natural, nenul) asociat, egal cu căți pași poate să sără spre dreapta, vom nota aceste numere cu $nr[0]$, $nr[1], \dots, nr[N-1]$. De exemplu dacă **blocul** e pe stiva cu indicele i , și numărul înscris pe el este $nr[bloc]$, atunci **se poate deplasa doar pe stiva $i+nr[bloc]$** .



În imagine se vede un exemplu de stare inițială. Opțiunile de răspuns însă se referă la orice caz de problemă (nu neapărat cu această stare inițială) decât dacă se precizează altfel.

$i+nr[bloc]$. Când un bloc este mutat peste o stivă cu alte blocuri pe ea, numărul posibil de pași pentru toate blocurile de pe stivă (inclusiv cel recent mutat) o să scadă cu 1. Nu este voie ca numărul înscris pe bloc să ajungă mai mic sau egal cu 0, deci o mutare care duce la o astfel de stare este invalidă.

Când un bloc este mutat pe ultima stivă, dispare (e scos din configurație), deci numărul total de blocuri scade cu 1.

Costul mutării unui bloc este numărul înscris la acel moment pe bloc (la ridicarea blocului de pe stivă, nu la așezarea lui, când deja pierde o unitate).

Scopul este ca toate blocurile să iasă din configurație (să ajungă pe ultima stivă).

- Starea inițială dată ca exemplu are soluție
- Pentru starea inițială, cu primul bloc cu numărul înscris mai mic decat numarul de stive, întotdeauna numărul de succesi este 1.
- Pentru orice stare numărul de succesi este mai mare sau egal cu 1.
- O stare inițială care conține doar blocuri cu numărul asociat 1 nu are soluție.
- O stare inițială care conține un bloc cu numărul asociat mai mare decât S nu are soluție.

Se consideră următoarea problemă a blocurilor modificată.

Bocurile alergătoare. (enunțul e identic cu cealaltă problemă numită "Blocuri alergătoare").

Avem S stive cu $S \geq 3$ și N blocuri, cu $N \geq 1$. În starea inițială, prima stivă (cea mai din stânga) conține toate blocurile. Fiecare bloc are un număr (natural, nenul) asociat, egal cu câți pași poate să sară spre dreapta, vom nota aceste numere cu $nr[0]$, $nr[1], \dots, nr[N-1]$. De exemplu dacă **blocul** e pe stiva cu indicele i, și numărul înscris pe el este $nr[bloc]$, atunci **se poate deplasa doar pe stiva $i+nr[bloc]$** . Când un bloc



În imagine se vede un exemplu de stare inițială. Opțiunile de răspuns însă se referă la orice caz de problemă (nu neapărat cu această stare inițială) decât dacă se precizează altfel. (15 puncte)

$i+nr[bloc]$. Când un bloc este mutat peste o stivă cu alte blocuri pe ea, numărul posibil de pași pentru toate blocurile de pe stivă (inclusiv cel recent mutat) o să scadă cu 1. Nu este voie ca numărul înscris pe bloc să ajungă mai mic sau egal cu 0, deci o mutare care duce la o astfel de stare este invalidă.

Când un bloc este mutat pe ultima stivă, dispare (e scos din configurație), deci numărul total de blocuri scade cu 1.
Costul mutării unui bloc este numărul înscris la acel moment pe bloc (la ridicarea blocului de pe stivă, nu la așezarea lui, când deja pierde o unitate).
Scopul este ca toate blocurile să iasă din configurație (să ajungă pe ultima stivă).

funcția $\hat{h}(\text{stare}) = (\text{numărul de blocuri existente în stare})$ este o estimare admisibilă pentru orice valori pentru N, S și pentru orice numere înscrise pe blocuri

funcția $\hat{h}(\text{stare}) = (\text{suma numerelor înscrise pe toate blocurile existente în stare})$ este o estimare admisibilă pentru orice valori pentru N, S și pentru orice numere înscrise pe blocuri

funcția $\hat{h}(\text{stare}) = (\text{maximul dintre toate numerele înscrise în blocuri})$ este o estimare admisibilă pentru orice valori pentru N, S și pentru orice numere înscrise pe blocuri

funcția $\hat{h}(\text{stare}) = (\text{minimul dintre toate numerele înscrise în blocuri})$ este o estimare admisibilă pentru orice valori pentru N, S și pentru orice numere înscrise pe blocuri

funcția $\hat{h}(\text{stare}) = (\text{suma distanțelor de la stiva fiecărui bloc la stiva finală})$ este o estimare admisibilă pentru orice valori pentru N, S și pentru orice numere înscrise pe blocuri. Se consideră distanța între stivele i și j, cu $i < j$, ca diferența $j-i$.

1

Care dintre afirmațiile de mai jos sunt adevărate pentru algoritmul Minimax (cu adâncime maxima setată)? **(15 puncte)**

- Rădăcina arborelui este întotdeauna un nod de tip MAX.
- Rădăcina arborelui este întotdeauna un nod de tip MIN.
- Rădăcina arborelui minimax poate fi și nod de tip MIN și nod de tip MAX
- Frunzele din arborele minimax sunt întotdeauna stări finale.
- Toate frunzele arborelui Minimax se găsesc mereu la aceeași adâncime în arbore.

2

Care dintre următoarele afirmații sunt adevărate pentru algoritmul Alpha-Beta (cu adâncime maxima setată)? **(15 puncte)**

- În orice arbore generat de Alpha-Beta se vor face retezări.
- Nu putem niciodată reteza primul fiu calculat al unui nod (presupunând că nodul nu e de asemenea retezat).
- Numărul maxim de retezări e egal cu numărul de fii ai rădăcinii.
- Dacă, pentru un nod N am avea în ordinea generării fiilor (mutările): f_1, f_2, \dots, f_n , dacă este retezat fiul f_i , cu $i < n$, atunci toți fiile de la f_i până la f_n sunt retezăți.

Important de citit. Convenții.

In subpunctele de mai jos se consideră pașii algoritmului aşa cum au fost predați la curs.

Se consideră convențiile de la curs cu privire la cine sunt MAX și MIN (calculator sau jucător).

Se consideră adâncimea maximă setată în toate problemele de mai jos ca fiind mai mare sau egală cu 1, unde adâncimea 0 e cea a rădăcinii iar adâncimea 1 e cea a filor rădăcinii. Pentru o adâncime maximă setată, ne oprim cu calculul fiilor la acea adâncime (nu vom genera nicun nod la adâncime strict mai mare decât ea).

Se consideră ordinea de evaluare a fraților în arborele Minimax de la stânga la dreapta.

3

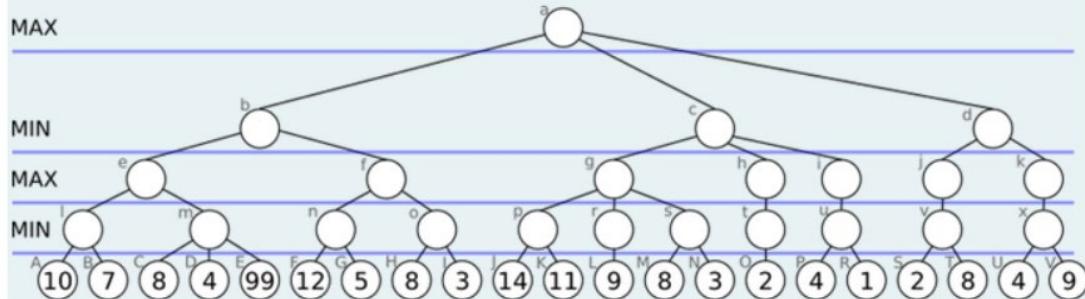
Care dintre afirmațiile de mai jos sunt adevărate pentru algoritmii Minimax și Alpha-Beta (cu adâncime maxima setată)? **(15 puncte)**

- Valoarea minimax a rădăcinii este sigur una dintre valorile frunzelor.
- Pentru o bună implementare a unui program care joacă un joc, folosind algoritmul minimax, în cadrul arborelui generat de algoritm, valoarea minimax asociată unui nod în care a câștigat calculatorul este mai mică decât valoarea unui nod în care a câștigat jucătorul uman.
- Eficiența Algoritmului Alpha-Beta depinde de ordinea în care sunt examinați succesorii.
- Cea mai potrivită valoare pentru o stare de remiză (stare finală în care nici calculatorul și nici jucătorul nu au câștigat) este 0.

4

Considerăm arborele Minimax din imagine pentru care cunoaștem valorile din frunze. Care dintre următoarele afirmații sunt adevărate?

(15 puncte)

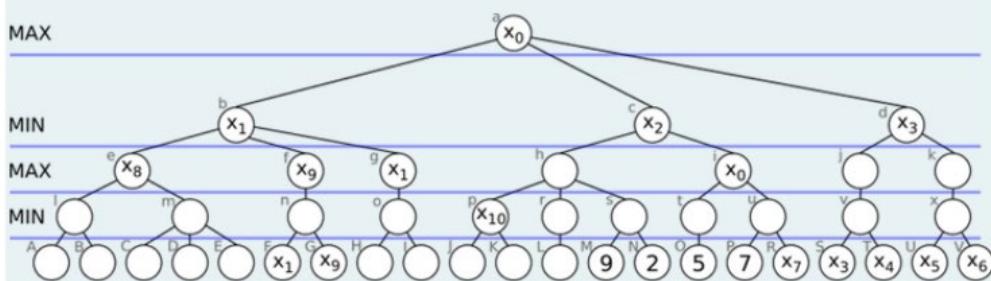


- Dacă am aplica algoritmul alpha-beta pe acest arbore și valoarea nodului G ar fi 55 în loc de 5, atunci nodul o nu ar mai fi evaluat.
- În nodul a vom avea valoarea minimax 99.
- Dacă am aplica algoritmul alpha-beta pe acest arbore, nodurile D și E nu ar mai fi evaluate.
- Aplicând algoritmul minimax, nodul h va avea în mod sigur valoarea 2.
- Dacă valorile frunzelor ar fi toate de 10 ori mai mari (pentru orice nod frunză, în loc de valoarea v am avea valoarea 10^*v) atunci setul de noduri din variația principală nu s-ar schimba.

1

Se consideră arborele minimax din imagine, cu adâncime maximă 4 (rădăcina fiind considerată la adâncime 0). Se presupune că arborele a fost deja calculat prin minimax, iar unele valori minimax au fost, apoi, fie sterse din imagine (nodurile fără conținut), fie înlocuite cu identificatori x_i (două noduri cu același identificator x_i au valorile minimax egale, însă putem avea $x_i = x_j$ pentru $i \neq j$). Care dintre afirmațiile de mai jos sunt sigur adevărate având în vedere informația dată despre arbore? Observație: frunzele sunt notate cu litere mari iar nodurile interne cu litere mici.

(15 puncte)



- x_3 este sigur mai mic sau egal cu x_4, x_5, x_6
- x_0 aparține intervalului $[5, 7]$
- x_2 este sigur egal cu x_0
- x_1 este sigur egal cu x_9
- Dacă aplicăm algoritmul Alpha-Beta asupra acestui arbore și x_8 este mai mare decât x_9 , atunci nodul g (literă mică) va fi retezat
- Dacă x_{10} e mai mic decât 9, nodul N (N - literă mare) va fi retezat.

2

Se consideră următoarea problemă de căutare:

Avem un grid de dimensiune $N \times N$, cu N natural, $N > 3$. În grid sunt C cutii colorate. În total există K culori, numerotate cu numere de la 1 la K . Putem muta doar câte o cutie pe rând. Cutia se poate deplasa doar cu o poziție în direcțiile sus, jos, stânga, dreapta și doar pe o poziție liberă care nu are ca vecină imediată (pe linie sau pe coloană) o cutie de altă culoare față de cutia mutată. Această regulă se aplică inclusiv stării inițiale în care nu putem avea cutii de culori diferite imediat vecine pe linie sau coloană.

Două cutii se consideră vecine dacă sunt fie pe aceeași linie dar pe coloane consecutive, fie pe aceeași coloană, dar pe linii consecutive.

Scopul este să grupăm cutiile de culori identice, astfel încât pentru orice culoare c_l , dacă numărul de cutii de culoare c_l este mai mare decât 1, orice cutie de culoare c_l să fie vecină cu altă cutie de culoare c_l .

Aveți exemple de stare inițială și finală la https://drive.google.com/drive/u/0/folders/1eLBRBusjat7zRQETtGmKoaFJI_M096EX

Care dintre afirmațiile de mai jos sunt adevărate?

(15 puncte)

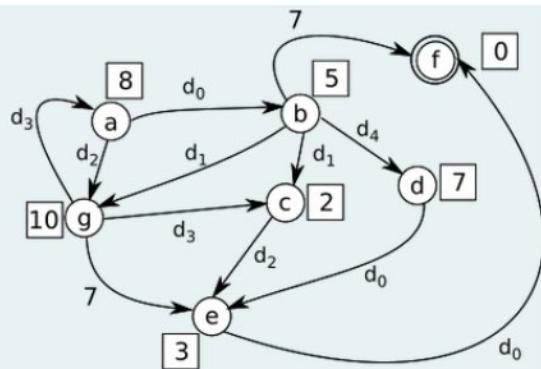
- Pentru $K=2$, dacă avem $N^2(N-1)/2$ cutii de culoare 1, dar strict mai mult de $(N-2)^2(N-1)$ cutii de culoare 2, atunci problema sigur nu are soluție, indiferent de configurația principală.
- Pentru $N=5$ numărul maxim de culori pentru ca problema să fie posibilă pentru că o configurație (să existe o stare inițială și o stare finală validă cu un număr de mutări natural nu neapărat nenul care duce de la starea inițială la cea finală) este 13.
- Numărul de succesi (valizi) ai fiecărei stări e suma numărului de locuri vecine (pe linie sau coloană) libere din jurul fiecărei cutii.
- Notăm $\text{dist_culoare}(cl) = \min(\{\text{multimea distanțelor Manhattan dintre toate plăcuțele de culoare } cl, \text{ nu neapărat distincte}\})$. O estimare admisibilă $\hat{h}(\text{nod}) = (\sum \text{valorilor dist_culoare}(cl) \text{ pentru toate culorile } cl)$.
- O estimare admisibilă $\hat{h}(\text{nod}) = \max(\{\text{multimea distanțelor dintre plăcuțele distincte de aceeași culoare}\})$.

4

Se dă graful orientat cu arce ponderate din imagine. Pentru unele arce ponderile nu sunt precizate, fiind înlocuite de identificatorii d_i . Euristică dată este una admisibilă (estimația pentru fiecare nod e trecută în pătrățelul de lângă nod).

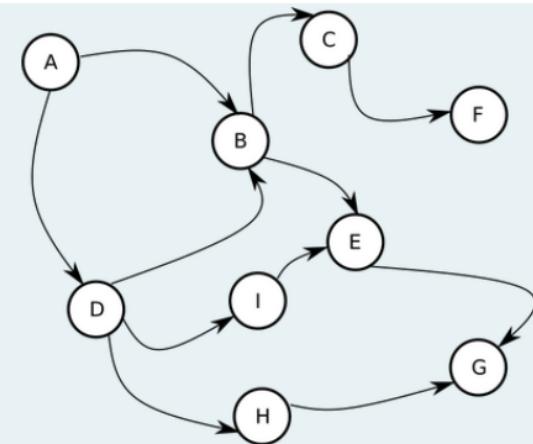
Costurile arcelor sunt numere naturale nenule. Drumul de cost minim returnat de A* pentru nodul de start a este $a \rightarrow b \rightarrow c \rightarrow e \rightarrow f$ cu costul 10.

Care dintre afirmațiile de mai jos sunt adevărate:
(15 puncte)



3

Se dă următorul graf orientat:
Care dintre următoarele afirmații sunt adevărate?
(15 puncte)



- Mulțimea {B,I} d-separă mulțimea {A, D} de mulțimea {C,E}.
- Graful dat nu este o topologie corectă pentru o rețea Bayesiană.
- Dacă am adăuga arcul F->A graful nu ar mai fi o topologie corectă pentru o rețea Bayesiană.
- Drumul de la nodul B la nodul F e blocat conditionat de mulțimea {C}
- Mulțimea {C,E} d-separă mulțimea {B,I,H} de mulțimea {F,G}

10

Care dintre afirmațiile de mai jos sunt adevărate? (15 puncte)

- d_0 are valoarea 3
- d_1 este mai mare strict decât 1
- d_2 este 1
- d_0 are valoare mai mare sau egală cu 3
- d_3 este 8
- pentru nodul de start a, nu mai există un drum în graf de același cost cu drumul de cost minim (reformulare, nu există două drumuri de cost minim pentru nodul de start a)

- Un sistem expert, pe baza informațiilor primite, oferă întotdeauna răspunsuri în totalitate certe.
- Un sistem expert trebuie să poată genera explicația (demonstrația) pentru un răspuns dat.
- Metoda înlățuirii înapoi presupune o căutare a răspunsului pornind de la date în încercarea de a găsi scopul.
- Un sistem expert este întotdeauna proiectat pentru a răspunde la întrebări din orice domeniu.
- Un sistem expert poate funcționa și fără o bază de cunoștințe.

15

Se consideră următoarea problemă de căutare:

Nodurile grafului conțin numere naturale mai mari sau egale cu 10.

Avem arc de la un nod n_1 la un nod n_2 dacă:

- **n_2 se obține prin n_1 prin inversarea a două cifre vecine**, caz în care **costul e egal cu maximul dintre cele două cifre inverse**. De exemplu, dacă $n_1=12483$, dacă alegem cifrele 1 și 2, atunci putem avea arcul $12483 \rightarrow 21483$ (de cost 2). Există o singură excepție în generarea acestui tip de succesor, și anume nu e voie prin inversare să punem prima cifră a numărului 0. De exemplu pentru 1023 nu putem inversa 1 cu 0.

- **n_2 se obține prin extragerea din n_1 a două cifre de pe poziții consecutive, însumarea lor, iar dacă suma este în continuare o cifră, adăugarea cifrei în aceeași poziție de unde au fost extrase cele două cifre (dacă suma lor nu e tot o cifră, nu se poate obține un succesor în acest mod). Costul unei astfel de transformări este chiar suma**. De exemplu, dacă $n_1=12483$, dacă alegem cifrele 1 și 2, atunci putem avea arcul $12483 \rightarrow 3483$ (de cost 3), dar pentru același număr nu putem alege 4 și 8. Nu putem aplica acest mod de obținere a succesorilor pentru numere de 2 cifre deoarece am ieșit din domeniul valorilor posibile pentru noduri.

Scopul este să ajungem la numere care încep și se termină cu aceeași cifră, de exemplu, 1321.

Exemplu de drum valid (nu neapărat de cost minim): $151772 \rightarrow 16772 \rightarrow 7772 \rightarrow 7727$ (costul este $(5+1)+(1+6)+7 = 20$)

Care dintre afirmațiile de mai jos sunt adevărate? (15 puncte)

Există minim 90 de noduri inițiale pentru care nu avem soluție

Nu există noduri inițiale de 3 cifre pentru care să nu avem soluție.

Pentru orice număr cu minim 9 cifre nenule avem sigur soluție.

O estimare admisibilă $\hat{h}(\text{nod})$ este, pentru cazul în care avem 2 cifre egale în număr, $\hat{h}(\text{nod})=\min(d_1+d_2)$ pentru toate cifrele c care au o dublură în număr, unde d_1 este distanța (în poziții) de la capătul din stânga la cea mai apropiată instanță a lui c de el, iar d_2 este distanța de la capătul din dreapta la cea mai apropiată instanță a lui c de el.

O estimare admisibilă $\hat{h}(\text{nod})$ este, pentru cazul în care avem 2 cifre egale în număr, este $\hat{h}(\text{nod})=0$ dacă nodul este final și $\hat{h}(\text{nod})=\max(\text{prima cifră și ultima cifră})$

Pentru nodul de start 18494 drumul de cost minim are costul 16

Pentru numarul 10010 singura stare scop în care poate ajunge este 10001

Numărul de coloane cu simbol majoritar ale lui MAX - numărul de coloane cu simbol majoritar ale lui MIN

scorul lui MAX de până acum din care scădem scorul lui MIN de până acum

numărul de linii deschise ale lui MAX - numărul de linii deschise ale lui MIN. O linie deschisă e un set de 3 căsuțe vecine pe rând, coloană sau diagonală care conțin doar simboluri ale jucătorului pentru care se calculează linia deschisă

numărul de simboluri izolate (fără vecini) ale lui MIN + numărul de simboluri izolate (fără vecini) ale lui MAX

numărul de randuri și coloane deschise ale lui MAX / numărul de randuri și coloane deschise ale lui MIN + (numărul de diagonale deschise ale lui MAX) / (numărul de diagonale deschise ale lui MIN). O linie deschisă e un set de 3 căsuțe vecine pe rând, coloană sau diagonală care conțin doar simboluri ale jucătorului pentru care se calculează linia deschisă

Un joc se desfășoară pe un grid $N \times N$. Jucătorii joacă cu simbolurile x și 0. Jucătorul cu simbolul x mută primul. Mutarea constă în plasarea unui simbol pe tablă.

La fiecare **diagonală de 3 simboluri** S de același fel, jucătorul cu simbolul S primește **scorul 2**, iar la **fiecare linie, sau coloană cu 3 simboluri** de același fel, primește **un scor de 1**. Simbolurile din configurația câștigătoare dispar. Dacă printre-o mutare se obțin mai multe configurații câștigătoare se adună toate scorurile pentru ele și dispar toate simbolurile implicate.

Care dintre variantele de mai jos oferă o funcție de evaluare care să arate în mod corect cât de favorabilă este o stare pentru calculator (MAX), cu alte cuvinte să aibă o valoare mai mare pentru stări mai favorabile și mai mică pentru stări mai nefavorabile pentru **orice stare intermediară (nefinală)** a jocului aflată la adâncimea maximă în arborele minimax? (15 puncte)

Se consideră problema blocurilor modificată astfel:

Fiecare bloc are asociată o literă. Literele se pot repeta (pot exista 2 blocuri cu aceeași informație).

Mutări posibile:

- Putem muta un bloc din vârful unei stive doar în vârful unei alte stive. Pe lângă configurația inițială se dă și o listă de cuvinte.
- Putem lua (șterge) un bloc din configurație.

Vom considera **costul pe mutarea unui bloc** ca fiind numărul de ordine al stivei pe care este mutat. Stivele se numerotează de la 1, pornind de la stânga spre dreapta.

Eliminarea unui bloc costă 1.

Scopul este ca prin mutările blocurilor să se ajungă ca fiecare dintre aceste cuvinte să reprezinte conținutul stivelor, citite de jos în sus, iar în rest să nu existe blocuri neîncadrate în cuvinte.

Se dă o configurație inițială cu blocurile amestecate și o listă de siruri LS.

De exemplu pentru starea inițială din folderul: <https://drive.google.com/drive/u/0/folders/1ynotW8u6WiLmFHUKOXUk0TmtA5zUP8oh> și LS conținând sirurile "mac", "bau" și "ba" putem avea ca stare finală posibilă cealaltă imagine din folder (atenție, nu e singurul caz posibil de stare finală pentru acest exemplu).

Vom nota cu **prefix(stiva,sir)**=numarul de caractere din prefixul comun al stivei și al cuvantului (de exemplu dacă stiva este "mih" și cuvântul este "miau", atunci prefix("mih","miau")=2.

Vom nota cu **lungime(sir)** lungimea sirului, respectiv lungime(stiva) înălțimea stivei

Vom nota cu **stive[i]** stiva de pe poziția i, unde cel mai mic i posibil este 1.

Care dintre următoarele afirmații sunt adevărate?

(15 puncte)

- Pentru orice stare inițială împreună cu orice set de siruri există soluție
- O estimare admisibilă $\hat{h}(\text{nod})$ este numărul de blocuri din configurația din nodul curent din care scădem lungimile adunate ale sirurilor.
- Consideram sirMax ca fiind sirul de lungime maximă. Fie $np = \text{prefix}(\text{stive}[1], \text{srMax})$ atunci estimarea $\hat{h}(\text{nod}) = \text{lungime}(\text{srMax}) - np$ este admisibilă
- Consideram sirMin ca fiind sirul de lungime minimă. Fie $np = \text{prefix}(\text{stive}[1], \text{srMin})$ atunci estimarea $\hat{h}(\text{nod}) = \text{lungime}(\text{srMin}) - np$ este admisibilă
- Notăm cu NS numărul de siruri din LS, și cu NGATA numărul de siruri (din LS) deja formate în stive (un sir e format într-o stivă dacă stiva conține acel sir).
Starea este fără blocuri suplimentare. Notăm NR=NS-NGATA. O estimare admisibilă $\hat{h}(\text{nod})$ este $NR * (NR + 1) / 2$

8

Pentru problema X și 0, care dintre următoarele afirmații sunt adevărate? Considerăm # simbolul pt loc liber (15 puncte)

Pentru starea inițială a tablei de joc, reprezentată prin $[['\#','\#','\#'], [\#,'\#','\#'], [\#,'\#','\#']]$, calculând arborele minimax în totalitate, fără a impune o adâncime maximă, numărul de noduri din arbore ar fi egal cu 9! (9 factorial = $1*2*3*4*5*6*7*8*9$).

O stare finală a jocului este ori una în care a câștigat MAX ori una în care a câștigat MIN.

Arborele minimax pentru X și 0 întotdeauna va avea un număr de niveluri mai mic sau egal cu 10.

Pentru starea curentă $[['x','\#','\#'], ['0','x','0'], ['x','\#','0']]$ (rădăcină a arborelui curent minimax), presupunând că simbolul calculatorului este X, arborele minimax de adâncime maximă 3 (în care numărul de muchii de pe un lanț de la rădăcină la un nod frunză nu poate depăși 3) are exact 10 noduri, inclusiv și rădăcina.

Considerând simbolul calculatorului ca fiind 0 (deci la începutul jocului utilizatorul mută primul) și numerotarea nivelurilor începând de la 0 (0 fiind nivelul rădăcinii), atunci pentru arborele minimax de adâncime maximă 4 având ca rădăcină tabla $[['\#','\#','\#'], ['0','x','\#'], ['x','\#','\#']]$, putem găsi o stare finală a jocului (câștig, pierdere sau remiză) pe nivelul 1 al arborelui.

9

Pentru o problemă oarecare (deci afirmația trebuie să fie adevărată pentru orice problemă de căutare cu costuri strict pozitive pe tranziții), care dintre formulele următoare ar obține din estimarea admisibilă $\hat{h}(\text{nod})$, oricare ar fi aceasta (adică formula trebuie să fie adevărată pentru orice \hat{h} nu pentru cazuri particulare de \hat{h}), o nouă estimare $\hat{h}_1(\text{nod})$ în mod cert neadmisibilă? Observație: în caz că nu se specifică nimic despre nodul dat ca parametru, înseamnă că se aplică formula pentru orice nod din graf, de orice tip. (15 puncte)

$\hat{h}_1(\text{nod})=\hat{h}(\text{nod})/2$

$\hat{h}_1(\text{nod})=\hat{h}(\text{nod})^2$

$\hat{h}_1(\text{nod})=\hat{h}(\text{nod})+2$, dacă nod nu e nod scop și $\hat{h}_1(\text{nod})=0$ dacă nod este scop

$\hat{h}_1(\text{nod})=\hat{h}(\text{nod})^3$ (cu sensul de ridicare la puterea a 3-a)

niciuna dintre formulele de la celelalte subpuncte ✓

KR LAB QUIZ seria 24 - LAB 1

Atenție, în exercițiile de mai jos se presupune că nu s-au suprascris numele implicate de funcții și metode.

De exemplu **NU** s-a scris o instrucțiune de genul

```
def sorted(l):  
    return l[::-1]
```

astfel încât sorted să nu mai sorteze ci să returneze lista inversată

Se consideră roulurile implicate ale funcțiilor și metodelor.

1

Se consideră (în python) variabila:

$I=[22,7,10,3,15,100]$

Care dintre următoarele afirmații sunt adevărate? (pentru fiecare afirmație se consideră valoarea inițială a variabilei I ca fiind cea din întrebare și nu valoarea schimbată de la vreun alt subiect)
(15 puncte)

- Nu există nicio funcție f astfel încât $I.sort(key=f)$ sa schimbe lista I în [10, 100, 15, 22, 3, 7]
- Putem defini o funcție $expr(x)$ astfel încât folosind expresia $I2=[expr(x) \text{ for } x \text{ in } I]$ să obținem în I2 lista [10,100]
- Există un sir s, astfel încât expresia $s.join(I)$ să aibă ca rezultat "22+7+10+3+15+100", fără a realiza alte operații asupra lui I în prealabil.
- Fie o listă I1. Expresia $\text{sorted}([(x,y) \text{ for } x \text{ in } I1 \text{ for } y \text{ in } I1])[::-1]==\text{sorted}([(y,x) \text{ for } x \text{ in } I1 \text{ for } y \text{ in } 2^*I], \text{reverse=True})$ este adevărată pentru orice listă I1, nevidă, de numere.
- Nu există nicio funcție cond(x) și nicio funcție expr(x) astfel încât în urma atribuirii $I1=[\text{expr}(x) \text{ for } x \text{ in } I \text{ if } \text{cond}(x)]$, I1 să fie [100, 15, 10]
- Pentru o listă I, de numere, nu neapărat cea din enunț expresia $I.sort()$ poate modifica variabila I, pe când expresia $\text{sorted}(I)$ nu îl modifică niciodată pe I.

2

Vom considera reprezentarea unei matrice de dimensiune $n*m$ ca fiind o listă de n liste diferite ca obiecte (nu avem două referințe de liste care să fie egale). Fiecare din cele n liste are m elemente.

Considerăm matricea $\text{matr}=[[1,4,8],[2,8,0],[5,1,9],[0,3,0]]$

Care dintre următoarele fraze sunt adevărate? **(15 puncte)**

- Pentru a obține numărul de zerouri din matricea matr putem scrie $\text{matr.count}(0)$
- $[[0]^n]^m$ creează o matrice validă cu n linii și m coloane și toate elementele nule.
- Să presupunem că mn este valoarea minimă dintre numărul de linii și numărul de coloane ale matricii matr, atunci expresia $[[\text{matr}[j][i] \text{ if } \text{matr}[j][i] > \text{matr}[i][j] \text{ else } \text{matr}[i][j]] \text{ for } i \text{ in range}(mn)] \text{ for } j \text{ in range}(mn)]$ va rezulta întotdeauna într-o matrice pătratică, simetrică față de diagonala principală.
- Dacă executăm instrucțiunea $\text{matr.append}(\text{matr}[0])$, iar apoi calculam $\text{matr}[0][0]=\max([\text{matr}[i+1][i] \text{ for } i \text{ in range}(\text{len}(\text{matr}[0]))] \text{ și } \text{matr}[-1][0]=\max([\text{matr}[i][0] \text{ for } i \text{ in range}(\text{len}(\text{matr}))])$, atunci prima și ultima linie din matricea matr vor fi mereu egale.
- Putem verifica egalitatea (adică au aceleași dimensiuni și aceleași elemente) a două matrici de numere, m1 și m2 prin m1==m2

3

Care dintre următoarele fraze sunt adevărate? **(15 puncte)**

- Pentru orice listă I, nevidă, există două perechi de valori (i,j) astfel încât $I[:i]+I[:j]$ să fie o listă palindrom (simetrică față de mijlocul său)
- Considerăm o listă I (de numere) pentru care facem operațiile: $\text{copie_I}=\text{list}(I)$, $I^*=n$, unde n e număr natural nenul și apoi $I.sort()$. Există un număr k astfel încât $I[:k]==\text{copie_I}$
- Fie o listă I nevidă și o variabilă $I2=I[a:b:c]$. Dacă I2 e nevidă atunci putem spune cu certitudine că $a < b$.
- Fie o listă I nevidă și o variabilă $I2=I[a:b:c]$. Dacă I2 are două elemente, în această ordine: e1 și e2, și c este negativ, atunci putem spune cu certitudine că, în lista I, e1 se află după e2.
- Pentru o lista $I2=I[a:b:c]$, cu c pozitiv, nenul, avem în I2 garantat parte întreagă din $|b-a|/c$ elemente, unde notația $|x|$ înseamnă modulul (valoarea absolută) lui x

4

Care din următoarele fraze sunt adevărate? **(15 puncte)**

- Pentru sirul $\text{Sir}="cotcodac si cip-cip-cip-cirip"$ există 6 numere: a,b,c,d,e,f astfel încât $\text{Sir}[a:b:c]+\text{Sir}[d:e:f]$ să fie egal cu "doctor-cip"
- Fie un sir de caractere nevid, s. Atunci putem spune că oricare ar fi i nenul, întreg, avem că $s[:i]$ e mereu egal cu $s[:i-1]+s[i:-1]$
- Pentru sirul $s="abcd"$, dacă scriem $s[0]=s[1]$, s va avea valoarea "aacd"
- Pentru sirul $\text{Sir}="pisica prinde soareci"$ există 6 numere: a,b,c,d,e,f astfel încât $\text{Sir}[a:b:c]+\text{Sir}[d:e:f]$ să fie egal cu "pisici aiici"

5

Se dă clasa de mai jos. Care dintre fraze sunt adevărate?

Observație: sunt 2 exerciții care folosesc aceeași clasă, deci nu e nevoie să recitați codul dacă ați mai făcut un exercițiu cu clasa curentă. (15 puncte)

```

77 class Pisica:
78     numar=0
79     soareciPrinsi=[]
80
81     def __init__(self, _nume, _varsta=0, _greutate=0):
82         self.nume=_nume
83         self.varsta=_varsta
84         self.greutate=_greutate
85         self.nrSoareci=0
86         self.__class__.numar+=1
87
88     def prindeSoareci(self, numeSoarece, greutateSoarece=0):
89         self.nrSoareci+=1
90         self.greutate+=greutateSoarece
91         self.soareciPrinsi.append(numeSoarece)
92
93     def __str__(self):
94         return "Pisica "+self.nume
95
96     def __repr__(self):
97         return "{} , {} a prins {} soareci".format(self.nume, self.varsta, self.nrSoareci)
98
99     @classmethod
100    def reseteazaSoareci(cls):
101        cls.soareciPrinsi=[]

```

- Creăm un obiect p1=Pisica("Mitzi", 10). În acest caz, în constructor, în parametrul self va intra valoarea Mitzi, în _nume 10, iar ceilalți 2 parametri vor avea valorile implicate 0 și 0.
- Constructorul clasei se poate apela fără argumente
- Numărul maxim de argumente pentru constructor este 4.
- Pentru p1=Pisica("Mitzi", _greutate=10) dacă afișăm, fără modificări intermediare, proprietatea varsta, aceasta va fi 0.
- pentru p1=Pisica("Mitzi", 10) un apel corect de metodă este p1.prindeSoareci(p1,"Chitzi",0.3)

6

Se dă clasa de mai jos. Care dintre fraze sunt adevărate?

Observație: sunt 2 exerciții care folosesc aceeași clasă, deci nu e nevoie să recitați codul dacă ați mai făcut un exercițiu cu clasa curentă. (15 puncte)

```

77 class Pisica:
78     numar=0
79     soareciPrinsi=[]
80
81     def __init__(self, _nume, _varsta=0, _greutate=0):
82         self.nume=_nume
83         self.varsta=_varsta
84         self.greutate=_greutate
85         self.nrSoareci=0
86         self.__class__.numar+=1
87
88     def prindeSoareci(self, numeSoarece, greutateSoarece=0):
89         self.nrSoareci+=1
90         self.greutate+=greutateSoarece
91         self.soareciPrinsi.append(numeSoarece)
92
93     def __str__(self):
94         return "Pisica "+self.nume
95
96     def __repr__(self):
97         return "{} , {} a prins {} soareci".format(self.nume, self.varsta, self.nrSoareci)
98
99     @classmethod
100    def reseteazaSoareci(cls):
101        cls.soareciPrinsi=[]

```

- pentru p1=Pisica("Mitzi", 5, 10), presupunând că nu s-au făcut modificări asupra variabilei, instrucțiunea print(p1) va afișa Pisica Mitzi
- pentru p1=Pisica("Mitzi", 5, 10), presupunând că nu s-au făcut modificări asupra variabilei, instrucțiunea print(p1) va afișa Mitzi, 5 a prins 0 soareci
- Dacă avem în program printre alte instrucțiuni, codul p2=Pisica("Pisi", 1, 5); p2.prindeSoareci("Chitzi"), imediat după aceste instrucțiuni avem garantia că p2.soareciPrinsi este lista ["Chitzi"]
Dacă avem în program printre alte instrucțiuni, codul p2=Pisica("Pisi", 1, 5); p2.soareciPrinsi=[]; p2.prindeSoareci("Chitzi"), iar după aceste instrucțiuni apelăm Pisica.reseteazaSoareci(), atunci în p2.soareciPrinsi nu vom avea elemente.
- Dacă încercăm să comparăm două instanțe ale clasei Pisica, p1=Pisica("Mitzi", 5, 10) și p2=Pisica("Pisi", 1, 5) prin p1<p2, se va arunca o eroare, deoarece nu este definită metoda __lt__() în cadrul clasei Pisica

7

Care dintre următoarele afirmații sunt adevărate? (15 puncte)

- Pentru o matrice m, salvată ca listă de liste, putem evalua dacă în interiorul său se află un element nul prin expresia `all([all(linie) for linie in m])`. Dacă expresia e True, înseamnă că nu avem elemente nule, și False în caz contrar.
- Pentru orice listă L nevidă, conținând valori ce pot fi evaluate ca booleene, expresia `not any([not x for x in L])==all(L)`
- Pentru o listă oarecare L, dacă e adevărat `all(L)` atunci sigur e adevărat și `any(L)`
- Fie o variabilă sirDat care conține un sir de caractere și un alt sir mai mic, numit separator. Expressia `all(sirDat.split(separator))` este adevărată dacă și numai dacă sirul separator nu apare în mod consecutiv ca subșir în sirul sirDat (de exemplu, sirul separator "#*" apare în mod consecutiv în "a#*#*bc" dar nu și în "a#*bc#*def")
- Considerăm o matrice m, de numere, reprezentată ca listă de liste. Presupunând că am importat funcția `reduce` din `functools`, și presupunând că avem într-o variabilă x o valoare numerică, atunci `reduce(lambda x,b:x+b, [lin.count(x) for lin in m])` oferă numărul de apariții al valorii lui x în matricea m

8

Care din următoarele afirmații sunt adevărate (15 puncte)

- Un bloc `try` poate avea asociate mai multe blocuri `except`
- Pentru o funcție definită cu `def f(x:int, y:str) -> int`, apelul `f(2.5, 10)` va arunca o eroare
- Există o funcție f astfel încât `min([10,417,22,0,99], key=f)` pentru lista dată să returneze 417
- Pentru o matrice m definită ca listă de liste, `max(max(m))` returnează cel mai mare element din matrice
- Pentru o matrice m definită ca listă de liste, expresia `[(lin, linie.index(0)) for lin, linie in enumerate(m)]` calculează o listă cu toate coordonatele (lin,coloană) în care se află elementul 0 în matrice.

9

Care dintre următoarele afirmații sunt adevărate, cu privire la tehniciile de căutare? Considerăm o soluție ca fiind un drum de la nodul start la un nod scop. (15 puncte)

- Algoritmul BreadthFirst va întoarce întotdeauna o soluție, indiferent care e nodul start și care sunt nodurile scop din graf.
- Într-un graf neconex, pot exista noduri scop pentru care nu avem un drum soluție (pornind de la nodul start)
- Într-un graf orientat, pe care aplicăm o problemă de căutare, avem măcar un drum soluție **dacă și numai dacă** nodul start și minim un nod scop se găsesc în aceeași componentă tare conexă
- Dacă am folosi algoritmul BF ca să obținem toate soluțiile pentru o problemă de căutare, ultima soluție returnată ar fi de lungime maximă
- Dacă nodul start este nod izolat în graf, mulțimea soluțiilor este vidă, indiferent care este mulțimea nodurilor scop din graf.

10

Considerăm un graf orientat cu $n > 2$ noduri numerotate cu numere de k cifre, $k \geq 2$ (numărul va fi informația nodului), unde prima cifră nu poate fi 0. Vom nota informația unui nod cu `info(nod)`. Considerând că mulțimea arcelor A este mulțimea tuturor perechilor de noduri (i,j) (arcul este de la i la j) cu proprietatea că între `info(i)` și `info(j)` diferența e de o singură cifră (de exemplu, pentru $k=3$, există arc de la 247 la 347, deoarece între ele diferă doar prima cifră). Vom considera scoperile ca fiind nodurile cu numere având toate cifrele egale (de exemplu, pentru $k=3$, 777 este un nod scop). Considerăm lungimea unui drum ca fiind numărul de muchii din drum. Care dintre următoarele afirmații sunt adevărate? (15 puncte)

- Pentru o valoare k oarecare, numărul de succesor ai unui nod este mereu 9^k (9 la puterea k)
- Pentru orice k și orice nod din graf ales ca fiind nodul start, lungimea celui mai scurt drum este $k-1$
- Vom presupune ordinea de generare a succesorilor următoarea: cifrele se schimbă de la stânga la dreapta și valorile cu care se schimbă se iau în ordine crescătoare. De exemplu, pentru $k=3$ și nodul 205, succesorii în ordine ar fi: 105, 305, 405, ..., 905, 215, 225, ..., 295, 201, 202, 203, 204, 206, 207, 208, 209. Atunci pentru orice k, dacă setăm nodul de start 10^{k-1} (10 la puterea k-1) drumul returnat de BreadthFirst este același cu drumul returnat de DepthFirst
- Pentru un k oarecare, dacă alegem drept nod start un nod scop, lungimea primului drum returnat de BreadthFirst este mereu k.
- Pentru un nod start cu $k-1$ cifre identice și o cifră diferită, primul drum returnat de BreadthFirst are în mod cert lungime 1, indiferent de ordinea evaluării succesorilor.
- Pentru un nod start cu $k-1$ cifre identice și o cifră diferită, primul drum returnat de DepthFirst are în mod cert lungime 1, indiferent de ordinea evaluării succesorilor.

LAB 2

1

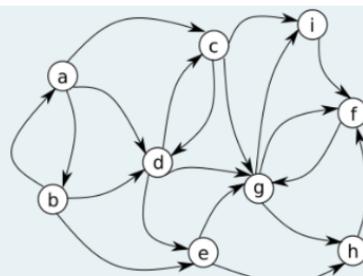
Considerăm graful din imaginea alăturată.

Vom considera că ordinea în care sunt evaluați succesorii pentru fiecare nod este ordinea alfabetică a informațiilor și ca dorim afișarea tuturor soluțiilor.

Considerăm lungimea unui drum ca fiind numărul de muchii din drum.

Consideram în cazul BF că intai se scoate din coada nodul de expandat și apoi se adaugă nodurile succesoare. De asemenea, capatul din stanga al cozii este cel de stergere și cel din dreapta cel de adaugare.

Care dintre următoarele afirmații sunt adevărate? (15 puncte)



- Nu există niciun nod start astfel încât, în cazul algoritmului BreadthFirst, coada de noduri să cuprindă noduri cu informație: d,e,c,d (scrise aici în ordinea nodurilor din coadă).
- Dacă nodul *a* e nod start și toate nodurile conținând o vocală sunt noduri scop, atunci lungimea, primului drum returnat de BreadthFirst e 2.
- Dacă nodul *a* e nod start și toate nodurile conținând o vocală sunt noduri scop, atunci lungimea, primului drum returnat de BreadthFirst e 0.
- În cazul în care nodul start este *a* și nodurile scop sunt *b* și *d*, primul drum soluție calculat de BreadthFirst e egal cu primul drum soluție calculat de DepthFirst
- În cazul în care nodul start este *a* și nodurile scop sunt *e* și *h*, primul drum soluție calculat de BreadthFirst e egal cu primul drum soluție calculat de DepthFirst

3

Care dintre următoarele fraze sunt adevărate: (15 puncte)

- Presupunem că avem o problemă care poate fi abstractizată la un graf. Aplicarea unei tehnici de căutare asupra problemei în scopul găsirii tuturor drumurilor soluție este echivalentă cu parcurgerea grafului (prin vizitarea și evaluarea unei singure date a tuturor nodurilor grafului, pe tot parcursul algoritmului)
- Problema creării orarului pentru profesorii dintr-o școală poate fi rezolvată folosind o tehnică de căutare considerând nodul initial ca fiind tabelul de ore fără nicio lecție setată, iar reprezentarea unei stări ca fiind orele deja planificate până în momentul t al stării. O tranziție ar fi plasarea unei noi lecții în tabel.
- Problema stabilirii traseului unui curier care are de livrat pachete în mai multe locații, astfel încât să nu treacă de 2 ori prin aceeași locație, se poate determina cu ajutorul unei tehnici de căutare.
- Problema generării unui răspuns automat coerent, relevant și util (care nu se găsește într-o lista de răspunsuri prestabilite) în urma unui input în limbaj natural al unui utilizator se poate rezolva cu o tehnică de căutare.
- Funcția succesor aplicată unei stări x returnează toate nodurile din subarborele avându-l ca rădăcină pe x, din cadrul arborelui reprezentând spațiul de căutare.
- O problemă de căutare poate să aibă oricâte noduri scop.

2

Considerăm următoarea problemă de căutare:

Se citesc dintr-un fișier numerele P, G, B, B1.

Un țăran vrea să mute de pe malul stâng al unui râu pe malul drept P pisici, G gaini și B saci cu boabe.

Țăranul are la dispoziție o barcă aflată la început pe malul stâng. Țăranul e singurul care poate muta barca (nu se pot deplasa pisici, gaini sau saci cu boabe fără țăran).

Atunci când barca pleacă de pe un mal (deci pe acel mal nu mai este țăranul), dacă există și pisici și saci (intacți) cu boabe, acestea vor alege un sac cu boabe și-l vor zgropăna până se desără, transformându-se în sac cu boabe desfăcut (chiar dacă sunt mai multe pisici, vor alege un singur sac cu boabe). De asemenea, pisicile nu au voie să rămână mai multe decât dublul numărului de gaini pe mal, fără țăran, fiindcă le vor ataca. Scopul țăranului este să mute, P pisici G gaini și B1 < B saci intacți cu boabe pe malul drept.

Bifați răspunsurile cu contextele (variabilele globale sau proprietățile statice pe care le avem) și structurile care pot reprezenta starea problemei **cu tot necesarul de informații pentru a cunoaște complet starea și a genera succesiuni pentru ea, dar în același timp, starea să nu conțină informații redundante**. Considerăm elementele din stare ca fiind numerele la momentul de timp în care se găsește starea. (15 puncte)

Salvăm numere P,G în variabile globale pe care nu le mai schimbăm. Starea poate fi reprezentată, în mod eficient, printr-un tuplu (sau o listă) cu valorile (numar_pisici_mal_stang, numar_gaini_mal_stang, numar_saci_intacti_mal_stang, numar_tarani_mal_stang, total_saci_intacti), unde total_saci_intacti se referă la suma numărului de saci intacti de pe ambele maluri.

Salvăm numere P,G,B în variabile globale pe care nu le mai schimbăm. Starea poate fi reprezentată, în mod eficient, printr-un tuplu (sau o listă) cu valorile (numar_pisici_mal_stang, numar_gaini_mal_stang, numar_saci_intacti_mal_stang, numar_tarani_mal_stang)

Salvăm numere P,G,B în variabile globale pe care nu le mai schimbăm. Starea poate fi reprezentată, în mod eficient, printr-un tuplu (sau o listă) cu valorile (numar_pisici_mal_stang, numar_gaini_mal_stang, numar_saci_intacti_mal_stang, numar_saci_desirati_mal_stang, locatie_barca)

Salvăm numere P,G în variabile globale pe care nu le mai schimbăm. Starea poate fi reprezentată, în mod eficient, printr-un tuplu (sau o listă) cu valorile (numar_pisici_mal_drept, numar_gaini_mal_drept, numar_saci_intacti_mal_stang, numar_saci_intacti_mal_drept, locatie_barca)

Salvăm numere P,G,B în variabile globale pe care nu le mai schimbăm. Starea poate fi reprezentată, în mod eficient, printr-un tuplu (sau o listă) cu valorile (numar_pisici_mal_stang, numar_pisici_mal_drept, numar_gaini_mal_stang, numar_gaini_mal_drept, numar_saci_intacti_mal_stang, numar_saci_intacti_mal_drept, locatie_barca)

Salvăm numere P,G,B în variabile globale pe care nu le mai schimbăm. Starea poate fi reprezentată, în mod eficient, printr-un tuplu (sau o listă) cu valorile (numar_pisici_mal_stang, numar_gaini_mal_stang, numar_saci_intacti_mal_stang, numar_saci_intacti_mal_drept, numar_tarani_mal_stang, locatie_barca, total_saci_intacti)

Consideram problema canibalilor și misionarilor.

Pe malul unui râu se află N misionari, N canibali și o barcă cu M locuri (N și M numere naturale). Scopul e ca toti oamenii să treacă râul cu barca. În nicio locație (inclusiv barca) nu avem voie să avem mai mulți canibali decât misionari deoarece, în acest caz, canibalii îi vor ataca pe misionari (stare invalidă). Barca nu poate pleca vida.

Considerăm reprezentarea stării sub forma unui tuplu (`numar_misionari_mal_initial`, `numar_canibali_mal_initial`, `mal_current`), unde `mal_current` reprezintă malul pe care se află barca. Vom nota malul inițial cu 1 și final cu 0.

Considerăm o tranziție ca fiind o singură deplasare cu barca (de la un mal la altul). Întoarcerea bărcii ar fi o nouă tranziție.

Considerăm lungimea unei soluții (drum de stări de la nodul start până la scop) ca fiind numărul de muchii din drum.

Care dintre următoarele afirmații sunt adevărate? (15 puncte)

- Pentru $M \geq 4$, indiferent care ar fi $N \geq 1$, problema are întotdeauna soluție.
- Starea inițială a problemei pentru orice M și N are un număr de succesiuni egal cu $M*(M+1)/2$
- Pentru M și N egale, mai mari strict decât 1, cea mai scurtă soluție are lungime 5.
- Pentru M și N egale, mai mari strict decât 1, cea mai scurtă soluție are lungime 3.
- Pentru M și N egale, mai mari strict decât 1, nu se poate spune care e lungimea celei mai scurte soluții.
- Presupunând că $N=3$ și $M=2$, un succesor al stării inițiale este (2,2,0)
- Presupunând că $N=3$ și $M=2$, un succesor al stării inițiale este (3,2,1)

LAB 3

Observație: cand este precizat "drumul returnat de A*" ne referim mereu la un drum-soluție (care începe cu un nod start și se termină cu un nod scop)

1. Care dintre următoarele fraze sunt adevărate pentru o problemă de căutare oarecare, abstractizată la un graf cu muchii ponderate (cu ponderi pozitive, neneule)?

Considerăm lungimea unui drum ca fiind numărul de arce/muchii din acel drum.

Pentru algoritmul A* se consideră implementarea descrisă pe pași de la curs.
(15 puncte)

- Pentru o evaluare euristică admisibilă, algoritmul A* oferă ca soluție drumul cu numărul minim de noduri, de la nodul start la nodul scop.
- Algoritmul A* oferă întotdeauna ca soluție drumul de lungime minimă indiferent de costurile de pe muchii/arce și evaluarea euristică asociată nodurilor grafului.
- Algoritmul A* oferă întotdeauna ca soluție drumul de cost minim, indiferent de costurile de pe muchii/arce și evaluarea euristică asociată nodurilor grafului.
- Algoritmul A* oferă întotdeauna ca soluție drumul de cost minim indiferent de costurile de pe muchii/arce dacă evaluarea euristică pentru orice nod N este mai mică decât costul oricărui drum de la N la un nod scop.
- Algoritmul A* oferă întotdeauna ca soluție drumul de cost minim indiferent de costurile de pe muchii/arce dacă evaluarea euristică pentru orice nod N este mai mare decât costul oricărui drum de la N la un nod scop.

2. Care dintre următoarele fraze sunt adevărate pentru orice problemă de căutare, abstractizată la un graf cu muchii ponderate (cu ponderi pozitive, neneule)? **(15 puncte)**

- Algoritmul A* poate fi folosit pentru calcularea drumului de lungime maximă prin înmulțirea costului de pe fiecare arc cu -1, lăsând factorul euristic la fel. În acest caz se ia primul drum returnat de algoritmul modificat, considerând costul său ca fiind cel calculat de A*-modificat înmulțit cu -1.
 - Algoritmul A* poate fi folosit pentru calcularea drumului de lungime maximă, ordonând coada OPEN descrescător după priorități și luând mereu primul nod din coadă pentru evaluare.
 - În drumul returnat de A* nu pot exista mai multe noduri scop.
- Algoritmul A* poate fi folosit pentru calcularea drumului de lungime maximă prin înlocuirea costului c al fiecărui arc cu $1/c$, modificând și factorul euristic h pentru fiecare nod, astfel încât să fie $1/h$. În acest caz se ia primul drum returnat de algoritmul modificat, considerând costul său ca fiind cel calculat însumând costurile inițiale de pe arce/muchii.

3. Care dintre următoarele fraze referitoare la notațiile și formulele folosite în A* sunt adevărate? Notăm cu n un nod oarecare din graf. **(15 puncte)**

- $\hat{g}(n)$ este factorul euristic, $\hat{h}(n)$ este factorul de adâncime, iar funcția euristică de evaluare are formula $\hat{h}(n) = -\hat{g}(n) + \hat{f}(n)$
- $\hat{g}(n)$ este factorul de adâncime, $\hat{h}(n)$ este factorul euristic, iar funcția euristică de evaluare, $\hat{f}(n)$ este suma dintre $\hat{g}(n)$ și $\hat{h}(n)$
- Pentru o estimare $\hat{h}(n) \leq h(n)$ pentru orice nod n din componenta conexă a nodului start, A* returnează în mod cert drumul de cost minim.
- O estimare $\hat{h}(n)$ este considerată neadmisibilă, dacă $\hat{h}(n) < h(n)$
- O estimare $\hat{h}(n)$ este considerată neadmisibilă, dacă $\hat{h}(n) > h(n)$

4. Care dintre următoarele fraze sunt adevărate? **(15 puncte)**

- A* e o tehnică de căutare neinformată.
- A* e o tehnică de căutare informată.
- A* se poate aplica numai pe grafuri orientate.
- A* se poate aplica numai pe grafuri conexe.

- În cazul în care aplicăm A* pe un graf fără ponderi pe muchii, dacă considerăm costul unei muchii egal cu 1, și estimarea nodurilor este admisibilă, atunci A* calculează drumul de lungime minimă (dacă acesta există) între nodul start și unul dintre nodurile scop.

5. Care dintre următoarele fraze sunt adevărate?

Observație (necesară pentru unele subpuncte): Vom presupune că nodurile de cost egal sunt evaluate mereu după aceeași relație de ordine pe stări. De exemplu, dacă relația de ordine pe stări este cea alfabetică, dacă în coadă avem nodurile b și c ambele cu costul estimat \hat{h} egal cu 5, b va fi înainte de c. De asemenea, presupunem că estimările pentru graful de dinainte de eventuala modificare (din fiecare subpunkt) sunt admisibile. (15 puncte)

Considerăm un graf neorientat oarecare cu un nod start și un singur nod scop z, iar soluția returnată de A* este $(a, n_1, n_2, \dots, n_k, z)$, unde n_1, n_2, \dots, n_k sunt noduri din graf. Dacă schimbăm problema, considerând

- nodul start ca fiind z, și unicul nod scop ca fiind a, și modificând estimările \hat{h} astfel încât să fie în continuare admisibile, atunci drumul returnat de A* este întotdeauna $(z, n_k, \dots, n_2, n_1, a)$.

Considerăm un graf neorientat oarecare cu un nod start și un singur nod scop z, iar soluția returnată de A* este $(a, n_1, n_2, \dots, n_k, z)$, unde n_1, n_2, \dots, n_k sunt noduri din graf. Dacă schimbăm problema, considerând

- nodul start ca fiind z, și unicul nod scop ca fiind n_i , unde n_i este unul dintre nodurile n_1, \dots, n_k , și modificând estimările \hat{h} astfel încât să fie în continuare admisibile, atunci drumul returnat de A* este întotdeauna (z, n_k, \dots, n_i) .

Considerăm un graf neorientat cu un nod start și cu mai multe noduri scop dintre care unul este z, iar

- soluția returnată de A* este $(a, n_1, n_2, \dots, n_k, z)$, unde n_1, n_2, \dots, n_k sunt noduri din graf. În acest caz putem spune cu certitudine că niciunul dintre nodurile n_1, n_2, \dots, n_k nu este nod scop.

Considerăm un graf neorientat oarecare cu un nod start și un singur nod scop z, iar soluția returnată de A* este $d=(a, n_1, n_2, \dots, n_k, z)$, unde n_1, n_2, \dots, n_k sunt noduri din graf. Dacă schimbăm problema, considerând

- nodul start, ca fiind z, și unicul nod scop ca fiind a, și modificând estimările \hat{h} astfel încât să fie în continuare admisibile, atunci drumul returnat de A* de la z la a, e întotdeauna de aceeași lungime cu d.

Considerăm un graf neorientat oarecare cu un nod start și un singur nod scop z, iar soluția returnată de A*

- este $d=(a, n_1, n_2, \dots, n_k, z)$, unde n_1, n_2, \dots, n_k sunt noduri din graf. Dacă schimbăm problema, considerând nodul start, ca fiind z, și unicul nod scop ca fiind a, dar lăsăm valorile estimărilor \hat{h} , atunci drumul returnat de A* de la z la a, e întotdeauna de aceeași lungime cu d.

6. Considerăm că graful problemei este un graf neorientat de tip arbore cu muchii ponderate și costuri numere naturale mai mari sau egale cu 1.

Vom presupune că nodurile de cost egal sunt evaluate mereu după aceeași relație de ordine pe stări atât în A* cât și în BreadthFirst.

Adâncimea unui nod n este numărul de muchii aflate în lanțul de la rădăcină la nodul n.

Se presupune că estimarea folosită de A* este admisibilă.

Bifați afirmațiile adevărate: (15 puncte)

Considerăm că nodul start e frunză și nodurile scop sunt de asemenea frunze (dar diferite de nodul start). Pentru

- fiecare legătură (muchie) $m=(n_1, n_2)$ din graf, alegem o funcție de cost $cost(m)=k*min_a+1$, unde min_a reprezintă minimul dintre adâncimile nodurilor și $k>2$. În această situație, primul drum soluție returnat de A* este și primul drum soluție returnat de BreadthFirst.

Considerăm că nodul start e rădăcina și nodurile scop sunt frunze. Pentru fiecare legătură (muchie)

- $m=(n_1, n_2)$ din graf, alegem o funcție de cost $cost(m)=k*min_a+1$, unde min_a reprezintă minimul dintre adâncimile nodurilor și $k>2$. În această situație, primul drum soluție returnat de A* este și primul drum soluție returnat de BreadthFirst.

Pentru fiecare legătură (muchie) $m=(n_1, n_2)$ din graf, alegem o funcție de cost $cost(m)=1$, atunci indiferent

- care este nodul start și care sunt nodurile scop, primul drum soluție returnat de A* este și primul drum soluție returnat de BreadthFirst (în cazul în care există un drum soluție).

Considerăm că nodul start e frunză și nodurile scop sunt de asemenea frunze (dar diferite de nodul start). Pentru

- fiecare legătură (muchie) $m=(n_1, n_2)$ din graf, alegem o funcție de cost $cost(m)=ad_max - max_a + 1$, unde max_a reprezintă maximul dintre adâncimile nodurilor n_1 și n_2 , iar ad_max este adâncime maximă a grafului. În această situație, primul drum soluție returnat de A* este și primul drum soluție returnat de BreadthFirst.

7. Considerăm că graful problemei este un graf oarecare, neorientat, de tip arbore având muchii cu ponderi numere naturale mai mari strict decât 1. (15 puncte)

- Dacă și nodul start și nodurile scop sunt frunze (diferite de nodul start), atunci, întotdeauna, nodul rădăcină din graful problemei va fi evaluat și expandat exact o singură dată de algoritmul A* până la oferirea soluției.

- Dacă și nodul start și nodurile scop sunt frunze (diferite de nodul start), atunci atât părintele nodului start, cât și părintele nodului scop (din soluție) vor fi întotdeauna evaluate și expandate exact o singură dată de algoritmul A* până la oferirea soluției.

- Dacă și nodul start și nodurile scop sunt frunze (diferite de nodul start), atunci fiecare nod existent în graful problemei va fi evaluat și expandat exact o singură dată de algoritmul A* până la oferirea soluției.

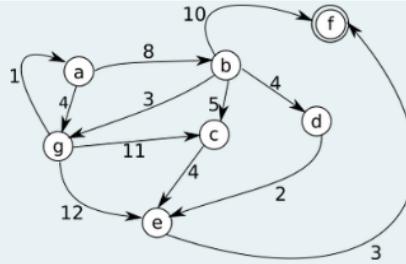
- Dacă nodul start este rădăcina și nodurile scop sunt frunze, drumul returnat de A* întotdeauna se va termina cu nodul scop aflat la cea mai mică adâncime.

LAB 4

1

Pentru graful din imagine avem următoarele stări ale cozii OPEN (ordonate). Prima stare e cea inițială a cozii open (cu nodul start inclus). Toate celelalte stări sunt afisate în urma repetării realizării acțiunii compuse din:

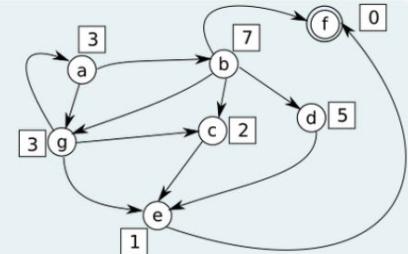
- 1) extragerea nodului cu f^* minim (nu se afiseaza coada pentru această stare intermedieră)
- 2) adăugarea succesorilor în coadă astfel încât coada să fie mereu ordonată crescător după f^* . (după această acțiune se afisează coada). Observație: Pentru valori f^* egale, elementele sunt în ordinea descrescătoare a valorii g . În cazul în care și f și g sunt egale, nodurile se ordonează după informație. Nodul start este a, iar nodul scop este f.



2

Pentru care dintre funcțiile de cost de mai jos (care asociază unui arc costul său), toate estimările, scrise în pătrătele, ale nodurilor din graful dat în imagine, sunt admisibile. Notăm cu n_1 și n_2 nodurile între care se găsește arcul $n_1 \rightarrow n_2$. Nodul start este a, nodul scop este f.

Am considerat: $\text{ord}(n)=\text{codul ASCII al literei nodului}$, $\text{abs}(\text{numar})$ returnează valoarea absolută a numărului, $\text{grad_intern}(n)$ returnează gradul intern al nodului (câte arce intră în nod). (15 puncte)



Stările cozii OPEN:

- [a]
- [b, g]
- [g, f, d, c]
- [c, f, d, e]
- [f, e, d]

Care dintre următoarele estimări (scrise între paranteze drepte pentru fiecare nod) ar fi în același timp admisibile și ar reprezenta și valori potrivite astfel încât coada OPEN a algoritmului A* să treacă prin valorile de mai sus?

(15 puncte)

- a[10] b[1] c[3] d[5] e[4] f[0] g[6]
- a[1] b[2] c[5] d[1] e[4] f[1] g[6]
- a[0] b[0] c[0] d[0] e[0] f[0] g[0]
- a[1] b[1] c[1] d[1] e[1] f[1] g[1]
- nu există un set de estimări astfel încât coada să treacă prin valorile date

4

Care dintre afirmațiile următoare sunt adevărate (notațiile sunt cele în prezentarea algoritmului A*): (15 puncte)

- În coada OPEN nu putem avea (în același timp) două noduri cu aceeași informație
- În coada OPEN nu vom găsi niciodată mai mult de un nod scop.
- În lista CLOSED nu vom găsi niciodată mai mult de un nod scop.
- estimarea nodului start nu poate fi niciodată 0.
- un nod se poate găsi și în OPEN și în CLOSED în același timp.
- Intotdeauna, la finalul executiei algoritmului toate nodurile din componenta conexă a nodului start se găsesc în CLOSED

5

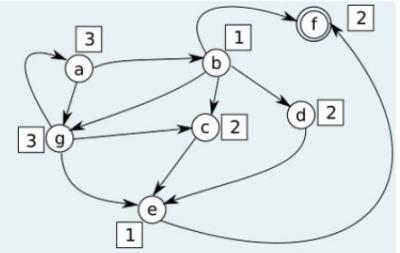
Pentru problema canibalilor și misionarilor cu N canibali, N misionari și M locuri în barcă, considerăm starea curentă $st=(n_{mi}, n_{ci}, b)$, unde n_{mi} reprezintă numărul de misionari aflați acum pe malul inițial, n_{ci} reprezintă numărul de canibali aflați curent pe malul inițial iar b e locația bărcii ($b=1$ pentru barca pe malul inițial și $b=0$ pentru barca pe malul final). Considerăm costul pe o mutare egal cu 1. Care dintre formulele pentru funcția de estimare duc la o funcție h admisibilă: (15 puncte)

- $\hat{h}(st)=\max(n_{ci}, n_{mi})/M$
- $\hat{h}(st)=\min(N-n_{ci}, N-n_{mi})$
- $\hat{h}(st)=\max(0, 2*(n_{mi}+n_{ci})/M - 1)$
- $\hat{h}(st)=2*(n_{mi}+n_{ci})/(M - 1) - 1 * b$
- $\hat{h}(st)=2*(n_{mi}+n_{ci})/(M - 1)$
- $\hat{h}(st)=1$
- $\hat{h}(st)=2*(\max(0, n_{mi}+n_{ci}-M))/(M - 1)+1 - 1 * b$

3

Pentru care dintre funcții de cost de mai jos (care asociază unui arc costul său), toate estimatiile, scrise în pătratele, ale nodurilor din graful dat în imagine, sunt admisibile. Notăm cu n_1 și n_2 nodurile între care se găsește muchia $n_1 \rightarrow n_2$. Nodul start este a, nodul scop este f.

Am considerat: $ord(nod)=$ codul ASCII al literei nodului, $abs(Numar)$ returnează valoarea absolută a numărului, $grad_intern(nod)$ returnează gradul intern al nodului (câte arce intră în nod), $sqr(nume)=$ radacina pătrată a numărului. (15 puncte)



- $cost(n_1, n_2)=1$, oricare ar fi n_1 și n_2
- $cost(n_1, n_2)=abs(ord(n_1)-ord(n_2))*10000$, unde $ord(n)=$ codul ASCII al literei nodului, iar $abs(Numar)$ returnează valoarea absolută
- $cost(n_1, n_2)=3$, oricare ar fi n_1 și n_2
- $cost(n_1, n_2)=ord(n_1)+ord(n_2)$
- $cost(n_1, n_2)=abs(grad_intern(n_1)-grad_intern(n_2))+sqrt(ord(n_1)+10*ord(n_2))$
- nu există o funcție de cost care să îndeplinească cerința problemei

LAB 5

Se consideră cunoscu enunțul clasical problemei blocurilor:

Aveam un număr N de blocuri/cuburi dispuse pe S stive. Se dă o stare initială, cu blocurile așezate, într-o anumită configurație, și una sau mai multe stări finale, cu blocurile rearanjate. Dorim să obținem mutările (lista ordonată de stări intermediare) prin care ajungem de la starea inițială la o stare finală.

Considerăm costul unei mutări egal cu 1.

1

Considerăm problema blocurilor cu următoarea modificare. Blocurile se pot muta doar pe o stivă imediat vecină (de pe stiva i putem muta un bloc doar pe $i+1$ sau $i-1$, dacă aceste stive există)

(15 puncte)

- Orice estimare admisibilă pentru problema blocurilor clasice este admisibilă și pentru această problemă.
- Numărul de succesiuni pentru orice stare este S^2
- Numărul de succesiuni pentru orice stare este S^2-2
- Dacă notăm cu S_v numărul de stive vide în starea curentă, numărul de succesiuni ai stării curente este $(S-S_v)^2-2$
- Dacă notăm cu S_v numărul de stive vide în starea curentă, numărul de succesiuni ai stării curente este $(S-S_v)^2(S-1)$

3

Pentru o problemă oarecare (deci afirmația trebuie să fie adevărată pentru orice problemă de căutare cu costuri strict pozitive pe tranziții), care dintre formulele următoare ar obține din **estimația admisibilă $\hat{h}(\text{nod})$** , mai mare sau egală cu 0, oricare ar fi aceasta (adică formula trebuie să fie adevărată pentru orice \hat{h} nu pentru cazuri particulare de \hat{h}), **o nouă estimație $\hat{h}_1(\text{nod})$ în mod cert admisibilă?** Observație: în caz că nu se specifică nimic despre nodul dat ca parametru, înseamnă că se aplică formula pentru orice nod din graf, de orice tip. (15 puncte)

- $\hat{h}_1(\text{nod}) = \hat{h}(\text{nod})/3$
- $\hat{h}_1(\text{nod}) = 1/\hat{h}(\text{nod})$
- $\hat{h}_1(\text{nod}) = \hat{h}(\text{nod})^4\hat{h}(\text{nod})$
- $\hat{h}_1(\text{nod}) = \min(\hat{h}(\text{nod})+2, \max(0, \hat{h}(\text{nod})-2))$
- Presupunând că mai avem încă o estimație admisibilă $\hat{h}_2(\text{nod})$, $\hat{h}_1(\text{nod}) = (\hat{h}(\text{nod}) + \hat{h}_2(\text{nod}))/2$

2

Considerăm problema blocurilor cu următoarea modificare. Nu putem să mutăm un bloc pe o stivă imediat vecină (nu pot muta de pe stiva i pe stiva $i+1$ sau $i-1$). Scopurile nu mai sunt date sub formă de configurații. Scopul este dat ca o condiție: și anume dorim să ajungem într-o configurație în care toate blocurile să fie adunate într-o singură stivă (oricare ar fi aceasta și oricare ar fi ordinea blocurilor.)

Care afirmații sunt adevărate? (15 puncte)

- Numărul stărilor scop este $N!$
- Numărul stărilor scop este $(N+1)!$
- Numărul stărilor scop este $S^N!$
- O estimație admisibilă pentru o stare cu S_v stive vide este $\hat{h}(\text{stare}) = \max(0, S-S_v-1)$
- O estimație admisibilă pentru o stare este $\hat{h}(\text{stare}) = \text{"numărul tuturor blocurilor care nu se găsesc pe una din stivele de înălțime maximă"}$
- Pentru o stare inițială cu $S=3$ și nicio stivă vidă, nu există drum soluție.

4

Se consideră următoarea problemă a blocurilor modificată.

Blocurile alergătoare.

(enunțul e identic cu cealaltă problemă numită "Blocuri alergătoare").

Aveți S stive cu $S \geq 3$ și N blocuri, cu $N \geq 1$. În stare initială, prima stivă (cea mai din stânga) conține toate blocurile. Fiecare bloc are un număr (natural, nenul) asociat, egal cu căți pași poate să sară spre dreapta, vom nota aceste numere cu $nr[0]$, $nr[1], \dots, nr[N-1]$. De exemplu dacă **blocul** este pe stivă cu indicele i , și numărul înscris pe el este $nr[bloc]$, atunci **se poate deplasa doar pe stiva $i+nr[bloc]$** . Când un bloc este mutat peste o stivă cu alte blocuri pe ea, numărul posibil de pași pentru toate blocurile de pe stivă (inclusiv cel recent mutat) o să scadă cu 1. Nu este voie ca numărul înscris pe bloc să ajungă mai mic sau egal cu 0, deci o mutare care duce la o astfel de stare este invalidă.

Când un bloc este mutat pe ultima stivă, dispare

(e scos din configurație), deci numărul total de blocuri scade cu 1.

Costul mutării unui bloc este numărul înscris la acel moment pe bloc (la ridicarea blocului de pe stivă, nu la așezarea lui, când deja pierde o unitate).

Scopul este ca toate blocurile să iasă din configurație (să ajungă pe ultima stivă).

În imagine se vede un exemplu de stare initială. Optiunile de răspuns însă se referă la orice caz de problemă (nu neapărat cu această stare initială) decât dacă se precizează altfel.

(15 puncte)



0 1 2 3 4 5 6 7 ieșire

5

Se consideră următoarea problemă a blocurilor modificată.

Blocurile alergătoare.

(enunțul e identic cu cealaltă problemă numită "Blocuri alergătoare").

Aveți S stive cu $S \geq 3$ și N blocuri, cu $N \geq 1$. În stare initială, prima stivă (cea mai din stânga) conține toate blocurile. Fiecare bloc are un număr (natural, nenul) asociat, egal cu căți pași poate să sară spre dreapta, vom nota aceste numere cu $nr[0]$, $nr[1], \dots, nr[N-1]$. De exemplu dacă **blocul** este pe stivă cu indicele i , și numărul înscris pe el este $nr[bloc]$, atunci **se poate deplasa doar pe stiva $i+nr[bloc]$** . Când un bloc este mutat peste o stivă cu alte blocuri pe ea, numărul posibil de pași pentru toate blocurile de pe stivă (inclusiv cel recent mutat) o să scadă cu 1. Nu este voie ca numărul înscris pe bloc să ajungă mai mic sau egal cu 0, deci o mutare care duce la o astfel de stare este invalidă.

Când un bloc este mutat pe ultima stivă, dispare

(e scos din configurație), deci numărul total de blocuri scade cu 1.

Costul mutării unui bloc este numărul înscris la acel moment pe bloc (la ridicarea blocului de pe stivă, nu la așezarea lui, când deja pierde o unitate).

Scopul este ca toate blocurile să iasă din configurație (să ajungă pe ultima stivă).

În imagine se vede un exemplu de stare initială. Optiunile de răspuns însă se referă la orice caz de problemă (nu neapărat cu această stare initială) decât dacă se precizează altfel. (15 puncte)



0 1 2 3 4 5 6 7 ieșire

Starea initială dată ca exemplu are soluție

Pentru starea initială, cu primul bloc cu numarul inscris mai mic decat numarul de stive, întotdeauna numărul de succesor este 1.

Pentru orice stare numărul de succesor este mai mare sau egal cu 1.

O stare initială care conține doar blocuri cu numărul asociat 1 nu are soluție.

O stare initială care conține un bloc cu numărul asociat mai mare decât S nu are soluție.

funcția $\hat{h}(\text{stare}) = (\text{numărul de blocuri existente în stare})$ este o estimare admisibilă pentru orice valori pentru N , S și pentru orice numere înscrise pe blocuri

funcția $\hat{h}(\text{stare}) = (\text{sumă numerelor înscrise pe toate blocurile existente în stare})$ este o estimare admisibilă pentru orice valori pentru N , S și pentru orice numere înscrise pe blocuri

funcția $\hat{h}(\text{stare}) = (\text{maximul dintre toate numerele înscrise în blocuri})$ este o estimare admisibilă pentru orice valori pentru N , S și pentru orice numere înscrise pe blocuri

funcția $\hat{h}(\text{stare}) = (\text{minimul dintre toate numerele înscrise în blocuri})$ este o estimare admisibilă pentru orice valori pentru N , S și pentru orice numere înscrise pe blocuri

funcția $\hat{h}(\text{stare}) = (\text{sumă distanțelor de la stiva fiecărui bloc la stiva finală})$ este o estimare admisibilă pentru orice valori pentru N , S și pentru orice numere înscrise pe blocuri. Se consideră distanța între stivele i și j , cu $i < j$, ca diferența $j-i$.

Important de citit. Convenții.

In subiectele de mai jos se consideră pașii algoritmului aşa cum au fost predăti la curs.

LAB 6

Se consideră convenția că MAX este calculatorul și MIN este utilizatorul.

Se consideră adâncimea maximă setată în toate problemele de mai jos ca fiind mai mare sau egală cu 1, unde adâncimea 0 e cea a rădăcinii iar adâncimea 1 e cea a fiilor rădăcinii. Pentru o adâncime maximă setată, ne oprim cu calculul fiilor la acea adâncime (nu vom genera nicun nod la adâncime strict mai mare decât ea).

Se consideră ordinea de evaluare a fraților în arborele Minimax de la stânga la dreapta.

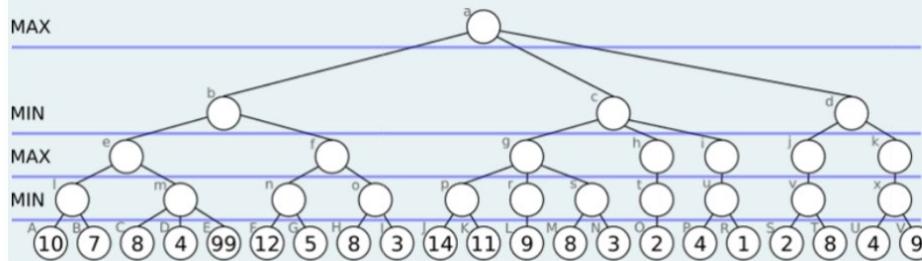
1

Care dintre estimările următoare sunt admisibile pentru problema 8-puzzle. Considerăm costul pe o mutare ca fiind numărul de plăcuțe vecine ale locației din care pleacă plăcuța mutată. De exemplu, dacă plăcuța era în colț și s-a mutat atunci avea 2 vecini (deoarece dintre cele 3 locații vecine una era sigur liberă). (15 puncte)

- $h(stare) =$ numărul de plăcuțe care nu sunt la locul lor
- $h(stare) = 2^*$ (numărul de plăcuțe care nu sunt la locul lor)
- $h(stare) = 3^*$ (numărul de plăcuțe care nu sunt la locul lor)
- considerăm $d(placuta) =$ distanța Manhattan dintre locația curentă a plăcuței și locația sa în starea scop.
 $h(stare) =$ suma tuturor valorilor $d(placuta)$ pentru toate plăcuțele din stare
- considerăm $d(placuta) =$ distanța Manhattan dintre locația curentă a plăcuței și locația sa în starea scop.
 $h(stare) = 2^*(\sum a_i)$ (a_i este numărul de plăcuțe care nu sunt la locul lor)

5

Considerăm arborele Minimax din imagine pentru care cunoaștem valorile din frunze. Care dintre următoarele afirmații sunt adevărate? (15 puncte)



- Dacă am aplica algoritmul alpha-beta pe acest arbore și valoarea nodului G ar fi 8 în loc de 5, atunci nodul o nu ar mai fi evaluat.
- În nodul a vom avea valoarea minimax 99.
- Dacă am aplica algoritmul alpha-beta pe acest arbore, nodurile D și E nu ar mai fi evaluate.
- Aplicând algoritmul minimax, nodul h va avea în mod sigur valoarea 2.
- Dacă valorile frunzelor ar fi toate de 10 ori mai mari (pentru orice nod frunză, în loc de valoarea v am avea valoarea 10^v) atunci setul de noduri din variația principală nu s-ar schimba.

2

Care dintre afirmațiile de mai jos sunt adevărate pentru algoritmul Minimax (cu adâncime maxima setată)? (15 puncte)

- Rădăcina arborelui are întotdeauna un fiu de tip MAX.
- Fie două noduri n_1 și n_2 frații, atunci fiile două noduri sunt ori toți de tip MIN ori toți de tip MAX (de exemplu, nu putem avea un nod de tip MAX fiu al lui n_1 și un nod de tip MIN fiu al lui n_2)
- Printre frunzele arborelui minimax pot fi și noduri de tip MIN și noduri de tip MAX în același timp
- Frunzele din arborele minimax sunt întotdeauna stări finale.
- Orice nod corespunzător unei stări finale a jocului se găsește întotdeauna la adâncime maximă în arborele minimax

3

Care dintre următoarele afirmații sunt adevărate pentru algoritmul Alpha-Beta (cu adâncime maxima setată)? (15 puncte)

- Pentru aceeași rădăcină (aceeași mutare pentru care calculăm arborei) și aceeași adâncime maximă setată, arboarele Minimax are un număr strict mai mare de noduri (în total) decât arborele Alpha-Beta
- Pentru un nod care nu este retezat, nu putem avea toți fiile lui retezăți
- Nu se va reteza niciodată un nod care e stare finală
- Dacă, pentru un nod N am avea în ordinea generării fiile (mutările): f_1, f_2, \dots, f_n , dacă este retezat fiul f_i , cu $i < n$, atunci toți fiile de la f_i până la f_n sunt retezăți.

4

Care dintre afirmațiile de mai jos sunt adevărate pentru algoritmii Minimax și Alpha-Beta (cu adâncime maxima setată)? (15 puncte)

- Valoarea minimax a rădăcinii este sigur una dintre valorile nodurilor aflate la adâncime maximă.
- Pentru o bună implementare a unui program care joacă un joc, folosind algoritmul minimax, în cadrul arborelui generat de algoritm, valoarea minimax asociată unui nod în care a câștigat calculatorul este mai mică decât valoarea unui nod în care a câștigat jucătorul uman.
- Eficiența Algoritmului Alpha-Beta depinde de ordinea în care sunt examinați succesorii.
- Cea mai potrivită valoare pentru o stare de remiză (stare finală în care nici calculatorul și nici jucătorul nu au câștigat) este 0.
- Pentru un fiu n , al rădăcinii, găsirea unui fiu n_1 al lui n , în care a câștigat MAX, garantează tăierea tuturor fraților lui n aflați în dreapta acestuia.

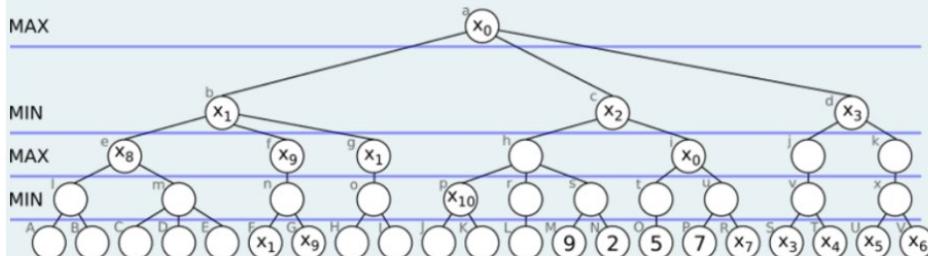
LAB 7

1

Se consideră arboarele minimax din imagine, cu adâncime maximă 4 (rădăcina fiind considerată la adâncime 0). Se presupune că arboarele a fost deja calculat prin minimax, iar unele valori minimax au fost, apoi, fie sterse din imagine (nodurile fără conținut), fie înlocuite cu identificatori x_i (două noduri cu același identificator x_i au valorile minimax egale, însă putem avea $x_i = x_j$ pentru $i \neq j$). Care dintre afirmațiile de mai jos sunt sigur adevărate având în vedere informația dată despre arbore?

Observație: frunzele sunt notate cu litere mari iar nodurile interne cu litere mici.

(15 puncte)



- x_3 este sigur mai mic sau egal cu x_4, x_5, x_6
- x_0 aparține intervalului $[5,7]$
- x_2 este sigur egal cu x_0
- x_1 este sigur egal cu x_9
- Dacă aplicăm algoritmul Alpha-Beta asupra acestui arbore și x_8 este mai mare decât x_9 , atunci nodul g (g-literă mică) va fi retezat.
- Dacă x_{10} e mai mic decât 9, nodul N (N - literă mare) va fi retezat.

2

Se consideră următoarea problemă de căutare:

Avem un grid de dimensiune $N \times N$, cu N natural, $N > 3$. În grid sunt C cutii colorate. În total există K culori, numerotate cu numere de la 1 la K .

Puteam muta doar câte o cutie pe rând. **Cutia se poate deplasa doar cu o poziție** în direcțiile sus, jos, stânga, dreapta și **doar pe o poziție liberă care nu are ca vecină imediată (pe linie sau pe coloană) o cutie de altă culoare față de cutia mutată**. Această regulă se aplică inclusiv stării inițiale în are nu putem avea cutii de culori diferite imediat vecine pe linie sau coloană.

Costul pe o mutare e egal cu 1.

Două cutii se consideră vecine dacă sunt fie pe aceeași linie dar pe coloane consecutive, fie pe aceeași coloană, dar pe linii consecutive.

Scopul este să grupăm cutiile de culori identice, astfel încât pentru orice culoare c_l , dacă numărul de cutii de culoare c_l este mai mare decât 1, orice cutie de culoare c_l să fie vecină (pe linie/coloană) cu altă cutie de culoare c_l .

Aveți exemple de stare inițială și finală la https://drive.google.com/drive/u/0/folders/1eLBRBujat7ZrOETtGmKoaFJI_M096EX

Care dintre afirmațiile de mai jos sunt adevărate?

(15 puncte)

- Pentru $K=2$, dacă avem $N*(N-1)/2$ cutii de culoare 1, dar strict mai mult de $(N-2)*(N-1)$ cutii de culoare 2, atunci problema sigur nu are soluție, indiferent de configurația principală.
- Pentru $N=5$ numărul maxim de culori pentru ca problema să fie posibilă pentru că o configurație (să existe o stare inițială și o stare finală validă cu un număr de mutări natural nu neapărat nenul care duce de la starea inițială la cea finală) este 13.
- Numărul de succesi (valizi) ai fiecărei stări e suma numărului de locuri vecine (pe linie sau coloană) libere din jurul fiecărei cutii.
- Notăm $\text{dist_culoare}(c_l) = \min(\{\text{multimea distanțelor Manhattan dintre toate plăcuțele de culoare } c_l, \text{ nu neapărat distincte}\})$. O estimare admisibilă $\hat{h}(\text{nod}) = (\sum \text{valorilor dist_culoare}(c_l) \text{ pentru toate culorile } c_l)$.
- O estimare admisibilă $\hat{h}(\text{nod}) = \max(\{\text{multimea distanțelor euclidiene dintre plăcuțele distincte de aceeași culoare}\})$.

3

Se consideră problema blocurilor, cu următoare modificări:

Pot exista mai multe blocuri cu aceeași informație (de exemplu, mai multe blocuri cu litera "a").

Un bloc oarecare **se poate muta** în două moduri:

- fie se mută de pe un vârf de stivă pe altul

- fie, dacă blocul se află la nivelul n și stiva din dreapta ori din stânga are înălțimea $n-1$, atunci blocul poate glisa în vârful stivei vecine, și toate blocurile care erau deasupra lui cad cu o poziție pe stivă respectivă.

Pentru ca o stare să fie considerată **scop** orice stivă din stare trebuie să conțină blocuri cu aceeași informație (de exemplu toate blocurile din stivă să aibă informația "a") și niciun bloc cu informația egală cu cele de pe stivă respectivă să nu se mai găsească pe alte stive (de exemplu, să nu fie două stive cu informația "a").

Costul pe o mutare este 1.

Notății:

- S = numărul de stive

- $SV(stare)$ = numărul de stive vide dintr-o stare

- N numărul de blocuri

- NR_INF - numărul de informații distincte din configurație (de exemplu dacă litera "a" apare de 3 ori în configurație, pentru ea se adună doar 1 în NR_INF).

Care din afirmațiile de mai jos sunt adevărate? (15 puncte)

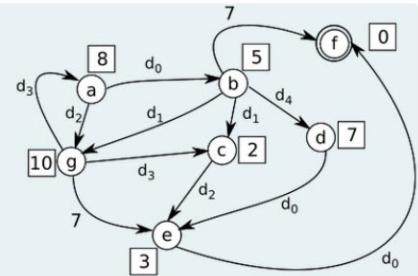
- Dacă $NR_INF > S$ atunci problema nu are soluție
- Pentru o stare în care orice stivă are proprietatea că stiva din dreapta are înălțimea strict mai mare decât a ei, numărul de succesor este $(S - SV(stare) + 1) * (S - 1)$
- Notăm cu $nr_st(info)$ numărul de stive distincte în care se găsesc blocuri cu informația info. O estimare admisibilă este $\hat{h}(stare) = \max(nr_st(info))$ pentru orice informație din configurație - 1.
- Notăm cu $\max(info)$ numărul maxim de blocuri cu aceeași informație din configurație. Notăm cu h_{max} înălțimea celei mai înalte stive. O estimare admisibilă este $\text{abs}(\hat{h}_{max} - \max(info))$, unde $\text{abs}(x)$ e valoarea absolută a lui x .
- Pentru o stare inițială în care problema are soluție, numărul total de stări scop posibile este $NR_INF!$ (factorial).
- Două stări se consideră distincte dacă există o pozitie i astfel încât informațiile de pe stiva de pe acea pozitie, din prima stare diferă ca ordine și/sau conținut față de informațiile de pe stiva de pe poziția i din cealaltă stare.

4

Se dă graful orientat cu arce ponderate din imagine. Pentru unele arce ponderile nu sunt precizate, fiind înlocuite de identificatorii d_i . Euristică dată este una admisibilă (estimarea pentru fiecare nod e trecută în pătrângelul de lângă nod).

Costurile arcelor sunt numere naturale nenule. Drumul de cost minim returnat de A* pentru nodul de start a este $a \rightarrow b \rightarrow c \rightarrow e \rightarrow f$ cu costul 10.

Care dintre afirmațiile de mai jos sunt adevărate: (15 puncte)



- d_0 are valoarea 3
- d_1 este mai mare strict decât 1
- d_2 este 1
- d_0 are valoare mai mare sau egală cu 3
- d_3 este 8
- pentru nodul de start a, nu mai există un drum în graf de același cost cu drumul de cost minim (reformulare, nu există două drumi de cost minim pentru nodul de start a)

Se consideră următoarea problemă de căutare:

Nodurile grafului conțin numere naturale mai mari sau egale cu 10.

Aveam arc de la un nod n_1 la un nod n_2 dacă:

- **n₂ se obține prin n₁ prin inversarea a două cifre vecine**, caz în care **costul e egal cu maximul dintre cele două cifre inversate**. De exemplu, dacă $n_1=12483$, dacă alegem cifrele 1 și 2, atunci putem avea arcul $12483 \rightarrow 21483$ (de cost 2). Există o singură excepție în generarea acestui tip de succesor, și anume nu e voie prin inversare să punem prima cifră a numărului 0. De exemplu pentru 1023 nu putem inversa 1 cu 0.
- **n₂ se obține prin extragerea din n₁ a două cifre de pe poziții consecutive, însumarea lor, iar dacă suma este în continuare o cifră, adăugarea cifrei în aceeași poziție de unde au fost extrase cele două cifre (dacă suma lor nu e tot o cifră, nu se poate obține un succesor în acest mod)**. Costul unei astfel de transformări este chiar suma. De exemplu, dacă $n_1=12483$, dacă alegem cifrele 1 și 2, atunci putem avea arcul $12483 \rightarrow 3483$ (de cost 3), dar pentru același număr nu putem alege 4 și 8. Nu putem aplica acest mod de obținere a succesorilor pentru numere de 2 cifre deoarece am ieșit din domeniul valorilor posibile pentru noduri.

Scopul este să ajungem la numere care încep și se termină cu aceeași cifră, de exemplu, 1321. Exemplu de drum valid (nu neapărat de cost minim): $151772 \rightarrow 16772 \rightarrow 7772 \rightarrow 7727$ (costul este $(5+1)+(1+6)+7 = 20$)

Care dintre afirmațiile de mai jos sunt adevărate? (15 puncte)

Există minim 90 de noduri inițiale pentru care nu avem soluție

Nu există noduri inițiale de 3 cifre pentru care să nu avem soluție.

Pentru orice număr cu minim 9 cifre nenule avem sigur soluție.

O estimare admisibilă $\hat{h}(nod)$ este, pentru cazul în care avem 2 cifre egale în număr, $\hat{h}(nod)=\min(d1+d2)$ pentru toate cifrele c care au o dublură în număr, unde d1 este distanța (în poziții) de la capătul din stânga la cea mai apropiată instanță a lui c de el, iar d2 este distanța de la capătul din dreapta la cea mai apropiată instanță a lui c de el.

O estimare admisibilă $\hat{h}(nod)$ este, pentru cazul în care avem 2 cifre egale în număr, este $\hat{h}(nod)=0$ dacă nodul este final și $\hat{h}(nod)=\max(\text{prima cifră și ultima cifră})$

Pentru nodul de start 18494 drumul de cost minim are costul 16

Pentru numarul 10010 singura stare scop în care poate ajunge e 10001

Un joc se desfășoară pe un grid $N \times N$. Jucătorii joacă cu simbolurile x și 0. Jucătorul cu simbolul x mută primul. Mutarea constă în plasarea unui simbol pe tablă.

La fiecare **diagonală de 3 simboluri** S de același fel, jucătorul cu simbolul S primește **scorul 2**, iar la **fiecare linie, sau coloană cu 3 simboluri** de același fel, primește **un scor de 1**. Simbolurile din configurația câștigătoare dispar. Dacă printr-o mutare se obțin mai multe configurații câștigătoare se adună toate scorurile pentru ele și dispar toate simbolurile implicate.

Care dintre variantele de mai jos oferă o funcție de evaluare care să arate în mod corect cât de favorabilă este o stare pentru calculator (MAX), cu alte cuvinte să aibă o valoare mai mare pentru stări mai favorabile și mai mică pentru stări mai nefavorabile pentru **orice stare intermedieră (nefinală)** a jocului aflată la adâncimea maximă în arborele minimax? (15 puncte)

Numărul de coloane cu simbol majoritar ale lui MAX - numărul de coloane cu simbol majoritar ale lui MIN

scorul lui MAX de până acum din care scădem scorul lui MIN de până acum

numărul de linii deschise ale lui MAX - numărul de linii deschise ale lui MIN. O linie deschisă e un set de 3 căsuțe vecine pe rând, coloană sau diagonală care conțin doar simboluri ale jucătorului pentru care se calculează linia deschisă

numărul de simboluri izolate (fără vecini) ale lui MIN + numărul de simboluri izolate (fără vecini) ale lui MAX

numărul de randuri și coloane deschise ale lui MAX / numărul de randuri și coloane deschise ale lui MIN + (numărul de diagonale deschise ale lui MAX) / (numărul de diagonale deschise ale lui MIN). O linie deschisă e un set de 3 căsuțe vecine pe rând, coloană sau diagonală care conțin doar simboluri ale jucătorului pentru care se calculează linia deschisă

Care dintre afirmațiile de mai jos sunt adevărate? (15 puncte)

Un sistem expert, pe baza informațiilor primite, oferă întotdeauna răspunsuri în totalitate certe.

Un sistem expert trebuie să poată genera explicația (demonstrația) pentru un răspuns dat.

Metoda înlănțuirii înapoi presupune o căutare a răspunsului pornind de la date în încercarea de a găsi scopul.

Un sistem expert este întotdeauna proiectat pentru a răspunde la întrebări în general din absolut orice domeniu (atotștiutor).

Un sistem expert poate funcționa și răspunde corect și fără o bază de cunoștințe.

7

Se consideră problema blocurilor modificată astfel:

Fiecare bloc are asociată o literă. Literele se pot repeta (pot exista 2 blocuri cu aceeași informație).

Mutări posibile:

- Putem muta un bloc din vârful unei stive doar în vârful unei alte stive.
- Putem lua (șterge) un bloc din configurație.

Pe lângă configurația initială se dă și o listă de cuvinte.

Vom considera **costul pe mutarea unui bloc** ca fiind numărul de ordine al stivei pe care este mutat. Stivele se numerotează de la 1, pornind de la stânga spre dreapta.

Eliminarea unui bloc costă 1.

Scopul este ca prin mutările blocurilor să se ajungă ca fiecare dintre aceste cuvinte să reprezinte conținutul stivelor, citite de jos în sus, iar în rest să nu existe blocuri neîncadrate în cuvinte.

Se dă o configurație inițială cu blocurile amestecate și o listă de siruri LS.

De exemplu pentru starea inițială din folderul: <https://drive.google.com/drive/u/0/folders/1ynotW8u6WiLmFHUKOXUk0TmtA5zUP8oh>

și LS conținând sirurile "mac", "bau" și "ba" putem avea ca stare finală posibilă cealaltă imagine din folder (atenție, nu e singurul caz posibil de stare finală pentru acest exemplu).

Vom nota cu **prefix(stiva,sir)**=numarul de caractere din prefixul comun al stivei și al cuvantului (de exemplu dacă stiva este "mih" și cuvântul este "miau", atunci $\text{prefix}(\text{"mih"}, \text{"miau"})=2$.

Vom nota cu **lungime(sir)** lungimea sirului, respectiv lungime(stiva) înălțimea stivei

Vom nota cu **stive[i]** stiva de pe poziția i, unde cel mai mic i posibil este 1.

Care dintre următoarele afirmații sunt adevărate?

(15 puncte)

- Pentru orice stare inițială împreună cu orice set de siruri există soluție
- O estimare admisibilă $\hat{h}(\text{nod})$ este numărul de blocuri din configurația din nodul curent din care scădem lungimile adunate ale sirurilor.
- Considerăm sirMax ca fiind sirul de lungime maximă. Fie $\text{np}=\text{prefix}(\text{stive}[1], \text{sirMax})$ atunci estimarea $\hat{h}(\text{nod})=\text{lungime}(\text{sirMax})-\text{np}$ este admisibilă
- Considerăm sirMin ca fiind sirul de lungime minimă. Fie $\text{np}=\text{prefix}(\text{stive}[1], \text{sirMin})$ atunci estimarea $\hat{h}(\text{nod})=\text{lungime}(\text{sirMin})-\text{np}$ este admisibilă
- Notăm cu NS numărul de siruri din LS, și cu NGATA numărul de siruri (din LS) deja formate în stive (un sir e format într-o stivă dacă stiva conține acel sir). Starea este fără blocuri suplimentare. Notăm $\text{NR}=NS-NGATA$. O estimare admisibilă $\hat{h}(\text{nod})$ este $\text{NR}^*(\text{NR}+1)/2$

8

Pentru problema X și 0, care dintre următoarele afirmații sunt adevărate? Considerăm # simbolul pt loc liber (15 puncte)

Pentru starea inițială a tablei de joc, reprezentată prin $[["#", "#", "#"], ["#", "#", "#"], ["#", "#", "#"]]$, calculând arborele minimax în totalitate, fără a impune o adâncime maximă, numărul de noduri din arbore ar fi egal cu $9!$ ($9 \text{ factorial} = 1*2*3*4*5*6*7*8*9$).

O stare finală a jocului este ori una în care a câștigat MAX ori una în care a câștigat MIN.

Arborele minimax pentru X și 0 întotdeauna va avea un număr de niveluri mai mic sau egal cu 10.

Pentru starea curentă $[["x", "#", "#"], [0, "x", "0"], ["x", "#", "0"]]$ (rădăcină a arborelui curent minimax), presupunând că simbolul calculatorului este X, arborele minimax de adâncime maximă 3 (în care numărul de muchii de pe un lanț de la rădăcină la un nod frunză nu poate depăși 3) are exact 10 noduri, incluzând și rădăcina.

Considerând simbolul calculatorului ca fiind 0 (deci la începutul jocului utilizatorul mută primul) și numerotarea nivelurilor începând de la 0 (0 fiind nivelul rădăcinii), atunci pentru arborele minimax de adâncime maximă 4 având ca rădăcină tabla $[["#", "#", "#"], [0, "x", "#"], ["x", "#", "#"]]$, putem găsi o stare finală a jocului (câștig, pierdere sau remiză) pe nivelul 1 al arborelui.

9

Pentru o problemă oarecare (deci afirmația trebuie să fie adevărată pentru orice problemă de căutare cu costuri strict pozitive pe tranzitii), care dintre formulele următoare ar obține din estimarea admisibilă $\hat{h}(\text{nod})$, oricare ar fi aceasta (adică formula trebuie să fie adevărată pentru orice \hat{h} nu pentru cazuri particulare de \hat{h}), o nouă estimare $\hat{h}_1(\text{nod})$ în mod cert neadmisibilă? Observație: în caz că nu se specifică nimic despre nodul dat ca parametru, înseamnă că se aplică formula pentru orice nod din graf, de orice tip. (15 puncte)

- $\hat{h}_1(\text{nod})=\hat{h}(\text{nod})/2$
- $\hat{h}_1(\text{nod})=\hat{h}(\text{nod})^2$
- $\hat{h}_1(\text{nod})=\hat{h}(\text{nod})+2$, dacă nod nu e nod scop și $\hat{h}_1(\text{nod})=0$ dacă nod este scop
- $\hat{h}_1(\text{nod})=\hat{h}(\text{nod})^3$ (cu sensul de ridicare la puterea a 3-a)
- niciuna dintre formulele de la celelalte subpuncte