CTI 362
Licu Mihai-George

Documentatie Tema Cautare Informata

exemplu rulare program:

python cautare_informata.py [INPUT_PATH] [OUTPUT_PATH] [NUM_SOL] [TIMEOUT]

python cautare_informata.py input output 2 30

python cautare_informata.py input output 1 5

Euristici:

"trivial" - cea banala, costul h va fi 1 daca nodul este scop altfel 0

"num_people" - admisibila, considera ca cost h numarul de oameni pe malul initial in starea respectiva, ca si cum ar trebui efectua n mutari pentru n oameni

"people_boat_capacity" - admisibila, considera numarul de mutari care trebuie efectuat considerand numarul total de oameni care mai trebuie mutat / numarul de locuri disponibile in barca (-1 daca barca trebuie sa si intoarca astfel mutand inapoi un om)

"bad" - inadmisibila, calculeaza diferenta dintre mancarea si canibalii de pe malul final

Fisiere de input:

small.txt - nu blocheaza nici un algoritm

big.txt - blocheaza algoritmii mai slabi

final.txt - starea initiala este si starea finala (mal initial = mal final)

no_solutions.txt - nu are solutii

Validari si optimizari:

Datele de intrare se testeaza prin functia valid_start care returneaza 0 daca cumva misionarii sunt depasiti numeric de canibali si nu exista destula mancare pentru a ii satisface pe acestia, sau daca numarul de calatorii minim necesar (nr_oameni /

nr_locuri_barca) este mai mic decat numarul de calatorii pe care barca le va putea suporta din cauza degradarii (nr_locuri_barca * degradare_barca).

Algoritmii sunt implementati cu structuri de date eficiente (de ex heapq pentru a*, greedy, ucs; deque pentru dfs, bfs)

Tabel:

input = big.txt (timeout 20s):

Algoritm	Euristica	Timp prima solutie (s*10^-2)	Cost f	Lungime
A*	bad	-	-	-
A*	num_people	72	13	14
A*	boat_capacity	-	-	-
A*	trivial	-	-	-
A* iterative deepening	bad	-	-	-
A* iterative deepening	num_people	12	13	14
A* iterative deepening	boat_capacity	-	-	-
A* iterative deepening	trivial	-	-	-
A* optimizat	bad	-	-	-
A* optimizat	num_people	1	13	14
A* optimizat	boat_capacity	-	-	-
A* optimizat	trivial	-	-	-
BFS	trivial	-	-	-
DFS	trivial	-	-	-
DFS iterative deepening	trivial	-	-	-
Greedy	bad	-	-	-
Greedy	num_people	10	11	12
Greedy	boat_capacity	128	17	18
Greedy	trivial	-	-	-
UCS	trivial	-	-	-

input = small.txt (timeout 20s):

Algoritm	Euristica	Timp prima solutie (s*10^-2)	Cost f	Lungime
A*	bad	500	5	6
A*	num_people	2	5	6
A*	boat_capacity	5	5	6
A*	trivial	399	5	6
A* iterative deepening	bad	-	-	-
A* iterative deepening	num_people	0.9	5	6
A* iterative deepening	boat_capacity	620	5	6
A* iterative deepening	trivial	-	-	-
A* optimizat	bad	-	-	-
A* optimizat	num_people	0.3	5	6
A* optimizat	boat_capacity	1548	7	8
A* optimizat	trivial	-	-	-
BFS	trivial	111	5	6
DFS	trivial	-	-	-
DFS iterative deepening	trivial	-	-	-
Greedy	bad	-	-	-
Greedy	num_people	1	5	6
Greedy	boat_capacity	0.5	5	6
Greedy	trivial	-	-	-
UCS	trivial	400	5	6

Subiecte abordate (verde) subiecte neabordate(rosu):

1. (5%)Fișierele de input vor fi într-un folder a cărui cale va fi dată în linia de comanda. În linia de comandă se va da și calea pentru un folder de output în care

programul va crea pentru fiecare fișier de input, fișierul sau fișierele cu rezultatele. Tot în linia de comandă se va da ca parametru și numărul de soluții de calculat (de exemplu, vrem primele NSOL=4 soluții returnate de fiecare algoritm). Ultimul parametru va fi timpul de timeout. Se va descrie în documentație forma în care se apelează programul, plus 1-2 exemple de apel.

2. (5%) Citirea din fisier + memorarea starii. Parsarea fișierului de input care respectă formatul cerut în enunț
3. (15%) Functia de generare a succesorilor
4. (5%) Calcularea costului pentru o mutare
5. (5%) Testarea ajungerii în starea scop (indicat ar fi printr-o funcție de testare a scopului). Atenție, acolo unde nu se precizează clar în fișierul de intrare o stare finală înseamnă că funcția de testare a scopului doar verifică niște condiții precizate în enunț. Nu se va rezolva generând toate stările finale posibile fiindca e ineficient, ci se va verifica daca o stare curentă se potrivește descrierii unei stări scop.
6. (15% = 2+5+5+3) 4 euristici:
    1. (2%) banala
    2. (5%+5%) doua euristici admisibile posibile (se va justifica la prezentare si in documentație de ce sunt admisibile)
    3. (3%) o euristică neadmisibilă (se va da un exemplu prin care se demonstrează că nu e admisibilă). Atenție, euristica neadmisibilă trebuie să depindă de stare (să se calculeze în funcție de valori care descriu starea pentru care e calculată euristica).
7. (10%) crearea a 4 fisiere de input cu urmatoarele proprietati:
    1. un fisier de input care nu are solutii
    2. un fisier de input care da o stare initiala care este si finala (daca acest lucru nu e realizabil pentru problema, aleasa, veti mentiona acest lucru, explicand si motivul).
    3. un fisier de input care nu blochează pe niciun algoritm și să aibă ca soluții drumuri lungime micuță (ca să fie ușor de urmărit), să zicem de lungime maxim 20.
    4. un fisier de input care să blocheze un algoritm la timeout, dar minim un alt algoritm să dea soluție (de exemplu se blochează DF-ul dacă soluțiile sunt cât mai "în dreapta" în arborele de parcurgere)
    5. dintre ultimele doua fisiere, cel putin un fisier sa dea drumul de cost minim pentru euristicile admisibile si un drum care nu e de cost minim pentru cea euristica neadmisibila
8. (15%) Pentru cele NSOL drumuri(soluții) returnate de fiecare algoritm (unde NSOL e numarul de soluții dat în linia de comandă) se va afișa:
    1. numărul de ordine al fiecărui nod din drum
    2. lungimea drumului
    3. costului drumului

4. timpul de găsire a unei soluții (**atenție**, pentru soluțiile de la a doua încolo timpul se consideră tot de la începutul execuției algoritmului și nu de la ultima soluție)
5. numărul maxim de noduri existente la un moment dat în memorie
6. numărul total de noduri calculate (totalul de succesori generati; atenție la DFI și IDA* se adună pentru fiecare iteratie chiar dacă se repetă generarea arborelui, nodurile se vor contoriza de fiecare dată afișându-se totalul pe toate iterațiile
7. între două soluții de va scrie un separator, sau soluțiile se vor scrie în fișiere diferite.

Obținerea soluțiilor se va face cu ajutorul fiecăruia dintre algoritmii studiați:

8. **Pentru studenții de la seria CTI problema se va rula cu algoritmii: BF, DF, DFI, UCS, A* (varianta care dă toate drumurile), A* optimizat (cu listele open și closed, care dă doar drumul de cost minim), IDA*.**

Pentru toate variantele de A* (cel care oferă toate drumurile, cel optimizat pentru o singură soluție, și IDA*) se va rezolva problema cu fiecare dintre euristici. Fiecare din algoritmi va fi rulat cu timeout, si se va opri daca depășește acel timeout (necesar în special pentru fișierul fără soluții unde ajunge să facă tot arborele, sau pentru DF în cazul soluțiilor aflate foarte în dreapta în arborele de parcurgere).

9. (5%) Afisarea in fisierele de output in formatul cerut
10. (5%+5%) Validări și optimizari. Veți implementa elementele de mai jos care se potrivesc cu varianta de temă alocată vouă:
    1. Validare: verificarea corectitudinii datelor de intrare
    2. Validare: găsirea unui mod de a realiza din starea initială că problema nu are soluții. Validările și optimizările se vor descrie pe scurt în documentație.
    3. Optimizare: găsirea unui mod de reprezentare a stării, cât mai eficient **(argumentabil eficient, de ex am folosit si left_can/right_can idem mis si food, si boat_deprecation + boat_seats + boat_counter cand se puteau calcula, doar pentru a imi fi mai usor in functii si afisari)**
    4. Optimizare: găsirea unor condiții din care sa reiasă că o stare nu are cum sa contina in subarborele de succesori o stare finala deci nu mai merita expandata (nu are cum să se ajungă prin starea respectivă la o stare scop).
    5. Optimizare: implementarea eficientă a algoritmilor cu care se rulează programul, folosind eventual module care oferă structuri de date performante.
11. (5%) Comentarii pentru clasele și funcțiile adăugate de voi în program (dacă folosiți scheletul de cod dat la laborator, nu e nevoie sa comentați și clasele existente). Comentariile pentru funcții trebuie să respecte un stil consacrat prin

care se precizează tipul și rolurile parametrilor, cât și valoarea returnată (de exemplu, reStructured text sau *Google python docstrings*).

12. (5%) Documentație cuprinzând explicarea euristicilor folosite. În cazul euristicilor admisibile, se va dovedi că sunt admisibile. În cazul euristicii neadmisibile, se va găsi un exemplu de stare dintr-un drum dat, pentru care h-ul estimat este mai mare decât h-ul real. Se va crea un tabel în documentație cuprinzând informațiile afișate pentru fiecare algoritm (lungimea și costul drumului, numărul maxim de noduri existente la un moment dat în memorie, numărul total de noduri). Pentru variantele de A* vor fi mai multe coloane în tabelul din documentație: câte o coloană pentru fiecare euristică. Tabelul va conține datele pentru minim 2 fișiere de input, printre care și fișierul de input care dă drum diferit pentru euristica neadmisibilă. În caz că nu se găsește cu euristica neadmisibilă un prim drum care să nu fie de cost minim, se acceptă și cazul în care cu euristica neadmisibilă se obțin drumurile în altă ordine decât crescătoare după cost, adică diferența să se vadă abia la drumul cu numărul K, K>1). Se va realiza sub tabel o comparație între algoritmi și soluțiile returnate, pe baza datelor din tabel, precizând și care algoritm e mai eficient în funcție de situație. Se vor indica pe baza tabelului ce dezavantaje are fiecare algoritm.
13. **Doar pentru studenții de la seria CTI (Bonus 10%) pentru implementarea Greedy și analizarea acestuia împreună cu ceilalți algoritmi.**
14. **Doar pentru studenții de la seria CTI (Bonus 10%) pentru fiecare dintre următoarele optimizări pentru cazul în care se cere o singură soluție (deci în program veți avea un if care verifică dacă numărul inițial de soluții cerut era 1):**

    1. **la BF să se returneze drumul imediat ce nodul scop a fost descoperit, și nu neapărat când ajunge primul în coadă**
    2. **la UCS+A* (bonusul ar fi 10%+10% dacă se face pt ambele) să nu avem duplicate ale informației din noduri în coadă. În cazul în care tocmai dorim să adăugăm un nod în coadă și vedem că există informația lui deja, păstram în coadă, dintre cele 2 noduri doar pe cel cu costul cel mai mic SI GREEDY**

15. **Bonus (5%) Dacă faceți mai multe euristici admisibile dar diferite ca idee (cu alt mod de abordare a calculării).**
16. **(Bonus 5%) implementare proprie (rescriere completă față de exemplele de la laborator)**
17. **(Bonus 15-20%) Adăugarea unei interfețe grafice care să simuleze drumul cu ajutorul unei animații (de exemplu, făcută în pygame).**