# ***Algoritmul de găsire de drum LPA\****

Dolete Anton, GRUPA 411-EIA

***Lifelong Planning A\**** = avand la baza algoritmul ***A\**** , ***LPA\**** refolosește informațiile din căutările anterioare.

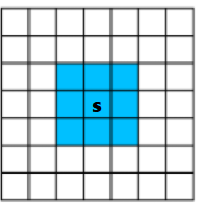
## *Abstractizare*

***Lifelong Planning A\* (LPA\*)***, în analogie cu „lifelong learning”, deoarece reutiliza informațiile din căutările anterioare. ***LPA\**** folosește euristica pentru a concentra căutarea și găsește întotdeauna calea cea mai scurtă pentru costurile marginale actuale. Prima căutare a ***LPA\**** este aceeași cu cea a lui ***A\*,*** dar toate căutările ulterioare sunt mult mai rapide. ***LPA\**** produce cel puțin arborele de căutare pe care ***A\**** îl construiește. Cu toate acestea, realizează o accelerare substanțială față de ***A\**** deoarece reutiliza acele părți ale arborelui de căutare anterior care sunt identice cu noul arbore de căutare.

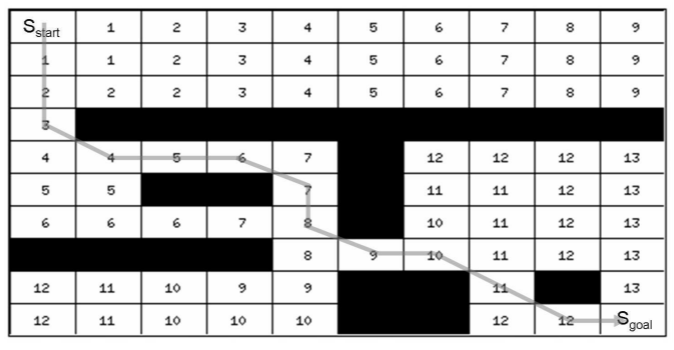
## *Descrierea funcţională a algoritmului LPA\**

Acest algoritm rezolvă următoarea problemă de căutare, aplicată unui ***graf*** a cărui ***cost de parcurgere*** creşte sau scade în timp. Denumim cu ***S setul de noduri ale grafului,*** formele anterioare ale grafului le denumim cu ***P(s),*** formele viitoare ale grafului cu ***V(s)*** şi costul de a ne muta de la punctul s la punctul ***s1*** cu ***c(s,s1***).

***LPA\**** va determina **cea mai scurtă cal**e de la nodurile start la finish(**cel mai mic cost** de parcurgere a grafului), ştiind topologia grafului şi costurile de parcurgere dintre noduri. O vizualizare simplă a costurilor de parcurgere ar fi împărţirea arealului într-o matrice conectată pe 8 direcţii, astfel că de la un nod de pornire numit s putem pleca în orice direcţie:



***Costul de deplasare*** în orice direcţie îl vom considera **1**; dacă în schimb una din căsuţe era imposibil de accesat şi parcurs, costul ei îl vom considera infinit. În figura de mai jos se observă punctul de start şi punctul ţintă; numerele din căsuţe reprezintă costul de a ajunge din punctul de start până la acea căsuţa ca fiind numărul de mutări.



În primă fază, ***LPA\**** porneşte asemeni cu algoritmul ***A\*.*** Îşi calculează costul ca fiind o sumă de distanţa parcursă până acum şi distanţa rămasă până la ţintă. Notăm cu **F costul cel mai mic dintre nodul curent şi nodul următor**,

F(n) = G(n)+RHS(n) unde

**G este costul mutării de la start la alt nod** iar

**RHS este euristica** (calea estimată) **pentru nodul curent când se schimbă arealul.**

Cu excepţia nodurilor de start şi de ţintă, fiecare nod are predecesor şi succesor. Orice nod a cărui colţ „indică” spre nodul X este un predecesor a lui X, orice nod mai aproape de ţintă decât X este un succesor a lui X.

Pentru fiecare nod ţinem cont de cele două costuri, G şi RHS. **Dacă G=RHS atunci nodul este consistent.** Dacă toate nodurile sunt consistente, LPA\* va determina calea cea mai scurtă în exact aceiaşi manieră precum A\*.

## *Rularea algoritmului*

Prima iteraţie a algoritmului este identică şi pentru LPA\* şi pentru A\*; în fond marele avantaj al LPA\* este evident doar la schimbarea arealului. Algoritmul începe prin setarea nodului de start ca având RHS = 0 şi G = infinit. Restul nodurilor au G = RHS = infinit până când sunt parcurse. În acest moment doar nodul de start se află în stiva de priorităţi căci este singurul neconsistent.

Valorile nodurilor sunt calculate(expandate) după cum urmează:

* Dacă G=RHS nodul este consistent şi va fi scos din stivă
* Dacă RHS < G, G este adus la valoarea lui H şi nodul este scos din stivă
* Dacă RHS > G, G este făcut infinit. Dacă după această operaţie nodul este consistent, el este scos din stivă. Dacă nu, cheia lui este recalculată.

Expandarea continuă cu următorul nod din vârful stivei până când se întrunesc simultan condiţiile:

* Nodul ţintă devine consistent
* Nodul din vârful stivei are o cheie mai mare sau egală decât cheia nodului ţintă

Nodurile sunt puse într-o **stivă de priorităţi**; prioritatea fiecărui nod se calculează după o cheie de sortare bidimensională:

k1(n) = [minim(g(n),rhs(n))+h(n,ţintă)] (corespunde valorilor din A\*)

k2(n) = minim(g(n),rhs(n))

După ce expandarea s-a terminat, este evaluată cea mai scurtă cale. Se începede la nodul ţintă şi pornim de la predecesorul nodului curent a cărui cost total este minim. Dacă mai multe noduri au acelaşi cost, oricare dintre ele reprezintă o cale de cost minim, putem alege oricare dintre ele. Repetăm acest pas până la nodul de start. Dacă costul este infinit, atunci nu există soluţie.

## Pseudocod LPA\*

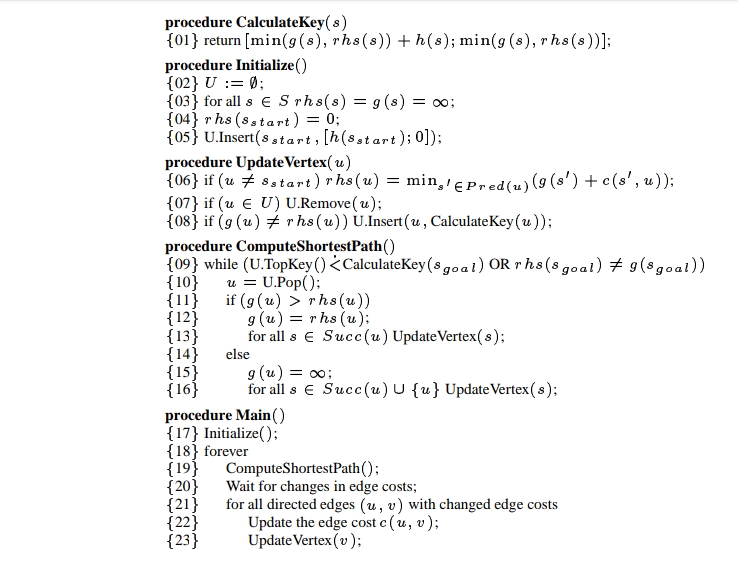


Figura 2 – Pseudocod LPA\*

## *Concluzii*

S-a demonstrat abilitatea LPA\* de a rezolva mai bine schimări în areal atunci când ele sunt minore, însă este nepotrivit pentru schimbări semnificative căci urmăreşte să expandeze nodurile apropiate de fosta traiectorie. Dezavantajul cerinţelor mari de memorie şi faptul că LPA\* trece de două ori prin fiecare nod(faţă de A\* care trece doar odată) şi implementarea dificilă sunt contraargumentele în folosirea acestui algoritm decât în situaţii speciale în care se aşteaptă schimbări mici în areal.

## *Surse şi bibliografie*

AI Magazine Volume 25 Number 2 (2004) (© AAAI): Incremental Heuristic Search in AI by Sven Koenig, Maxim Likhachev, Yaxin Liu, and David Furcy

<https://papers.nips.cc/paper/2001/file/a591024321c5e2bdbd23ed35f0574dde-Paper.pdf>

[/wiki/Lifelong\_Planning\_A\*](https://en.wikipedia.org/wiki/Lifelong_Planning_A*)

<https://dl.acm.org/doi/10.1016/j.artint.2003.12.001>

<https://www.slideserve.com/klaus/lifelong-planning-a>

<http://theory.stanford.edu/~amitp/GameProgramming/Variations.html>

[https://en.wikipedia.org/wiki/Lifelong\_Planning\_A\*](https://en.wikipedia.org/wiki/Lifelong_Planning_A*)

[Source code LPA\*](https://github.com/zhm-real/PathPlanning/tree/master/Search_based_Planning/Search_2D)

[CodeShare](https://collabedit.com/rr2q3)