OSNOVE UMETNE INTELIGENCE

2021/22

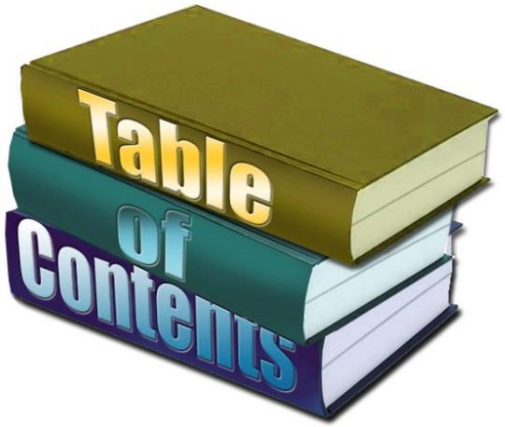
planiranje
razporejanje opravil

Pridobljeno znanje s prejšnjih predavanj

- **igranje iger med nasprotnikoma**
  - dva igralca, MIN in MAX, izmenične poteze v igralnem drevesu
  - igralca vplivata na vrednost **kriterijske funkcije** v listih
  - **algoritem MINIMAX** določa optimalno strategijo, če igralca igrata optimalno
  - **rezanje alfa-beta** vrne isto zaporedje potez kot bi algoritem MINIMAX s to razliko, da ne upošteva vej, ki ne vplivajo na končno odločitev
    - alfa, beta, prenašanje vrednosti v globino, posodabljanje vrednosti navzgor
    - rezultat rezanja odvisen od vrstnega reda vozlišč, časovna zahtevnost?

- **planiranje**
  - začetno stanje, akcije, ciljno stanje
  - akcije imajo: predpogoje, učinke, omejitve
  - jezika STRIPS, PDDL

Pregled



III. PLANIRANJE in razporejanje opravil

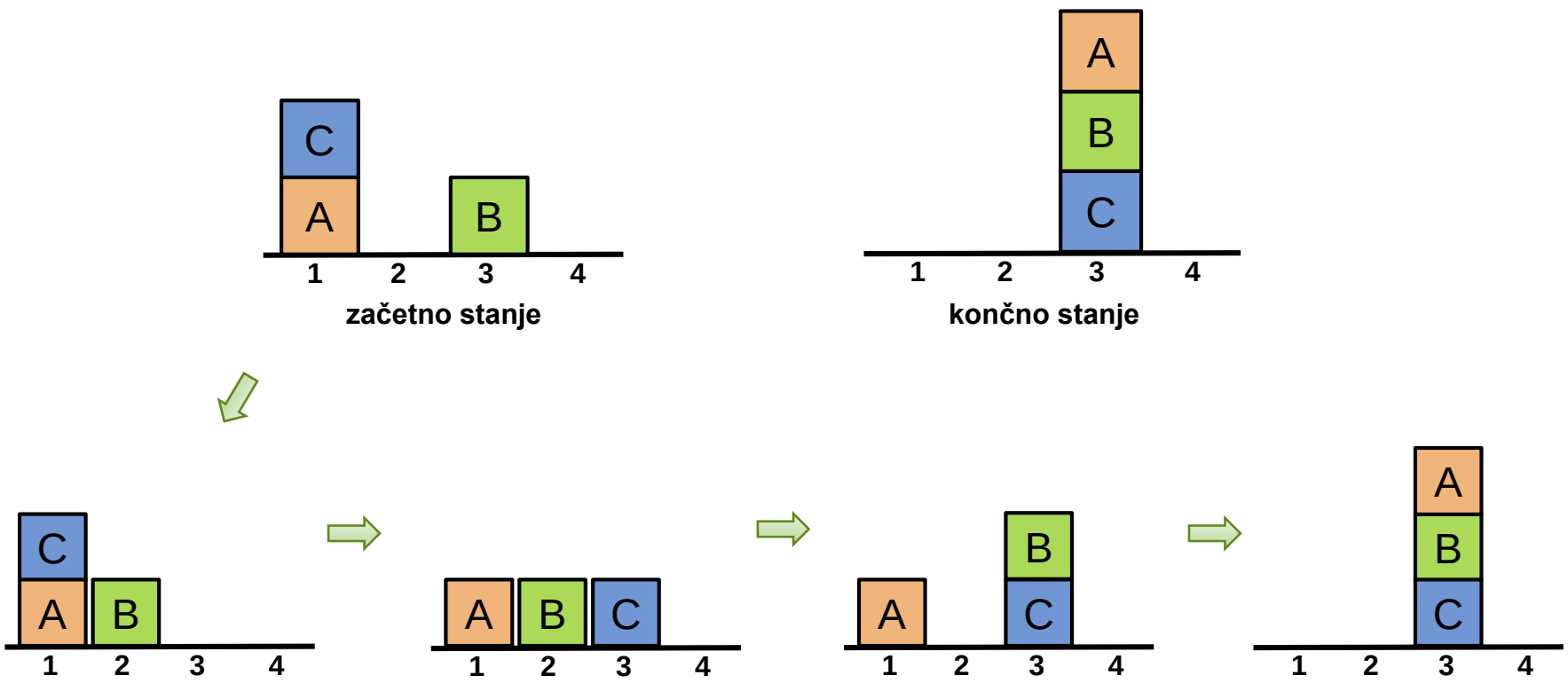
- predstavitev problema
- planiranje s "klasičnim" preiskovanjem prostora stanj
- planiranje s sredstvi in cilji
- planiranje z regresiranjem ciljev
- razporejanje opravil

Primer 1: Svet kock

Plan: zaporedje akcij, ki pripelje od začetnega do končnega stanja

Možna rešitev:

```
plan = [move(b,3,2), move(c,a,3), move(b,2,c), move(a,1,b)]
```



Primer 2: menjava pnevmatike (PDDL)

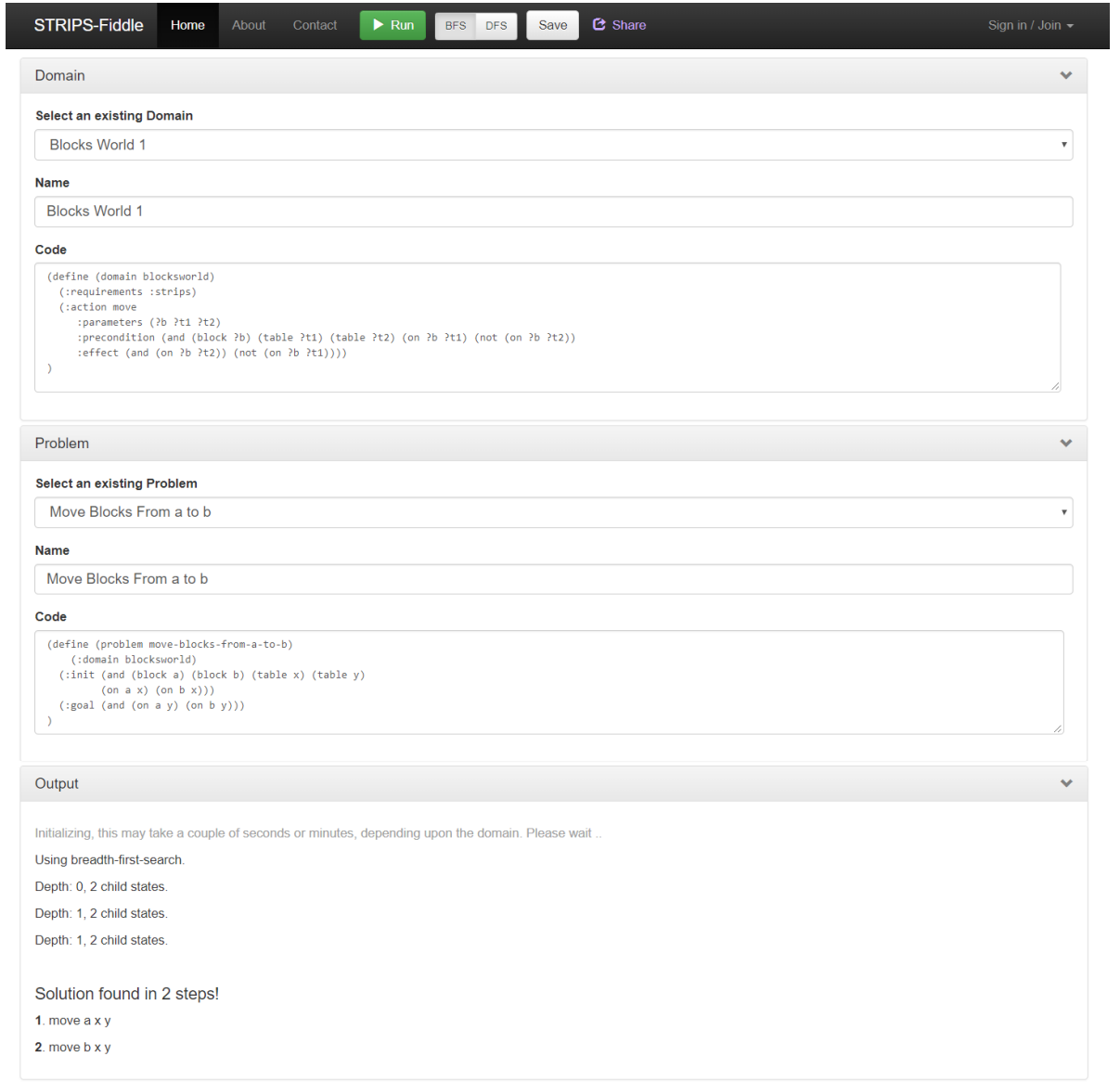
```
Init(Tire(Flat) ∧ Tire(Spare) ∧ At(Flat, Axle) ∧ At(Spare, Trunk))
Goal(At(Spare, Axle))
Action(Remove(obj, loc),
    PRECOND: At(obj, loc)
    EFFECT: ¬At(obj, loc) ∧ At(obj, Ground))
Action(PutOn(t, Axle),
    PRECOND: Tire(t) ∧ At(t, Ground) ∧ ¬At(Flat, Axle) ∧ ¬At(Spare, Axle)
    EFFECT: ¬At(t, Ground) ∧ At(t, Axle))
Action(LeaveOvernight,
    PRECOND:
    EFFECT: ¬At(Spare, Ground) ∧ ¬At(Spare, Axle) ∧ ¬At(Spare,Trunk)
    ∧ ¬At(Flat, Ground) ∧ ¬At(Flat, Axle) ∧ ¬At(Flat, Trunk))
```

- Možna rešitev?
- Kako se akcije odražajo na **zaporednih spremembah stanja**?

Demo

- https://stripsfiddle.herokuapp.com/

- primeri PDDL:
  - menjava avtomobilske gume (Frenk Dragar, 2020/21; Andraž Žnidar, 2021/22)
  - kuhanje kave (Matic Klopčič, 2021/22)

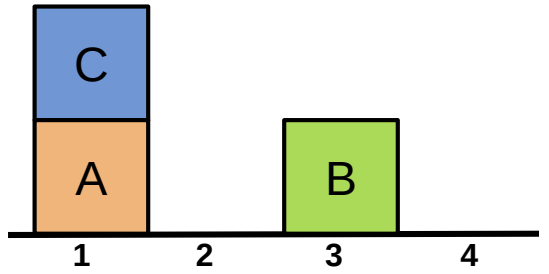


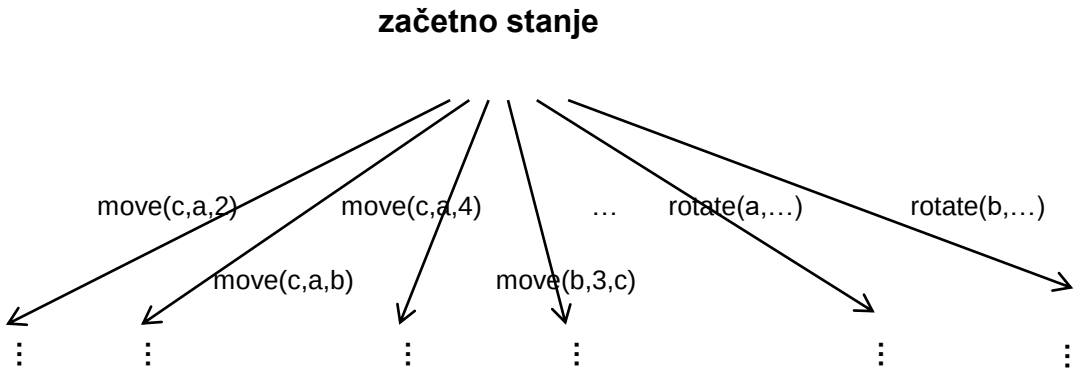
“Klasično” preiskovanje prostora stanj

- uporaba neinformiranih, informiranih ali lokalnih **preiskovalnih algoritmov**
- rezultat: **kombinatorična eksplozija** prostora stanj
- iskanje v smeri od začetnega k ciljnemu stanju lahko razvija vozlišča **z uporabo akcij, ki niso relevantne**

- **primeri**:
  - možni premiki kock iz začetnega stanja
  - akcija: `buy(Isbn)`

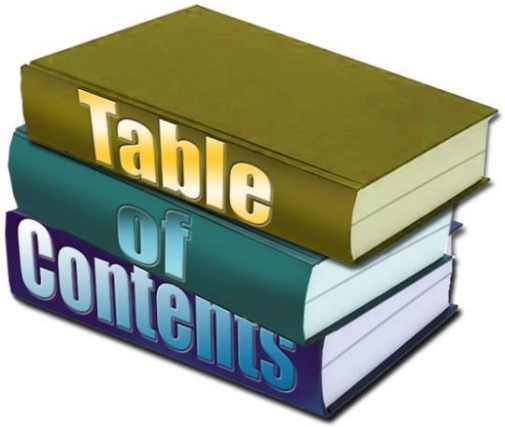
    začetno stanje: `[]`,
    predpogoji: `[]`,
    učinki: `add [own(Isbn)]`,
    cilj: `[own(1234567890)]`





- rešitve:
  - dobra **hevristična ocena**
  - **drugačen pristop** k preiskovanju

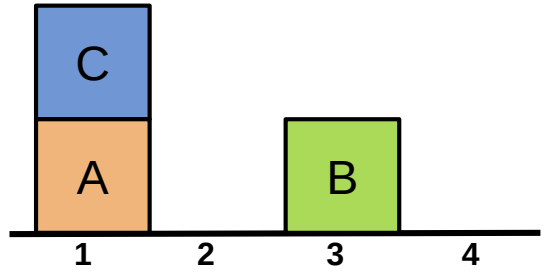
Pregled



III. PLANIRANJE in razporejanje opravil

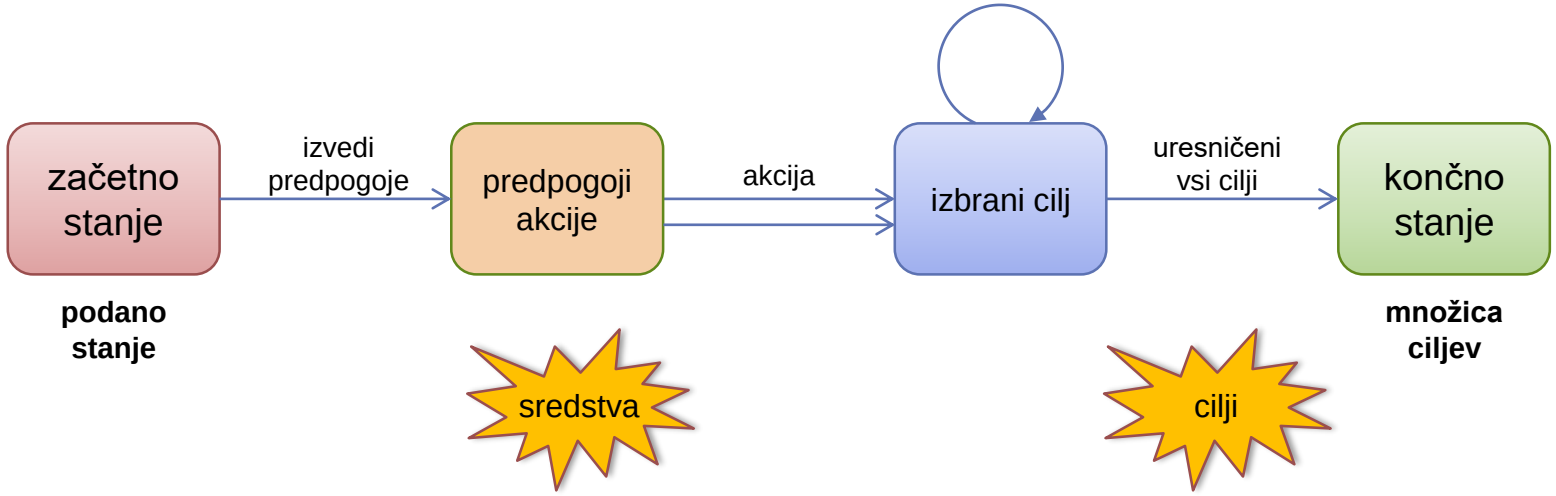
- predstavitev problema
- planiranje s "klasičnim" preiskovanjem prostora stanj
- planiranje s sredstvi in cilji
- planiranje z regresiranjem ciljev
- razporejanje opravil

Planiranje s sredstvi in cilji

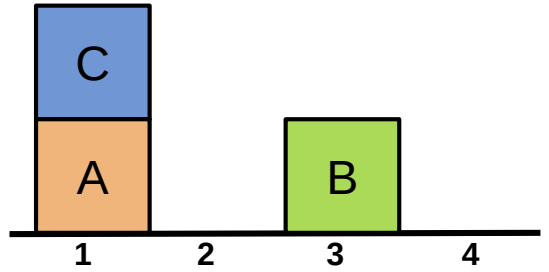


- primer iz sveta kock
  stanje: `[on(c,a), on(a,1), on(b,3), clear(c), clear(2), clear(b), clear(4)]`
  cilj: `[on(a,b),on(b,c)]`

- način reševanja:
  1. izberi **nerešen cilj**
  2. izberi **akcijo**, ki lahko vzpostavi (doseže) ta cilj
  3. ker ima akcija **predpogoje**, omogoči akcijo z **izvedbo predpogojev**
  4. **izvedi akcijo**
  5. vrni se v **korak 1** ali
     končaj, če so **uresničeni vsi cilji**



Planiranje s sredstvi in cilji



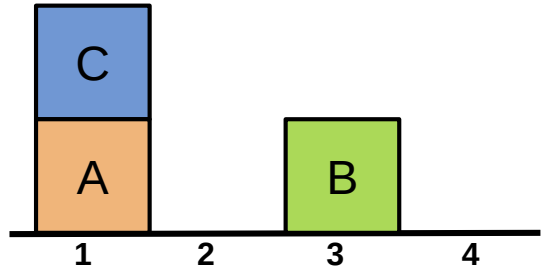
- primer iz sveta kock
  stanje: `[on(c,a), on(a,1), on(b,3), clear(c), clear(2), clear(b), clear(4)]`
  cilj: `[on(a,b),on(b,c)]`

- rešitev plana **pri preiskovanju v globino**:

```
move(c,a,2)   % izpolnjujemo on(a,b), ki potrebuje clear(a)
move(a,1,b)   % izpolnimo cilj on(a,b)
move(a,b,1)   % izpolnjujemo on(b,c), ki potrebuje clear(b), pokvarimo on(a,b)
move(b,3,c)   % izpolnimo cilj on(b,c)
move(a,1,b)   % ponovno izpolnimo cilj on(a,b)
<plan zaključen, vsi cilji izpolnjeni>
```

- pomembne podrobnosti:
  - strategija preiskovanja (v globino, širino, iterativno poglabljanje)
    - ali bi **iterativno poglabljanje** in **iskanje v širino** našla najkrajši plan?
  - princip **varovanja (ščitenja) ciljev** (angl. *goal protection*): pri preiskovanju lahko dodatno varujemo, da ne podremo že doseženih ciljev

Planiranje s sredstvi in cilji



- primer iz sveta kock
  stanje: `[on(c,a), on(a,1), on(b,3), clear(c), clear(2), clear(b), clear(4)]`
  cilj: `[on(a,b),on(b,c)]`

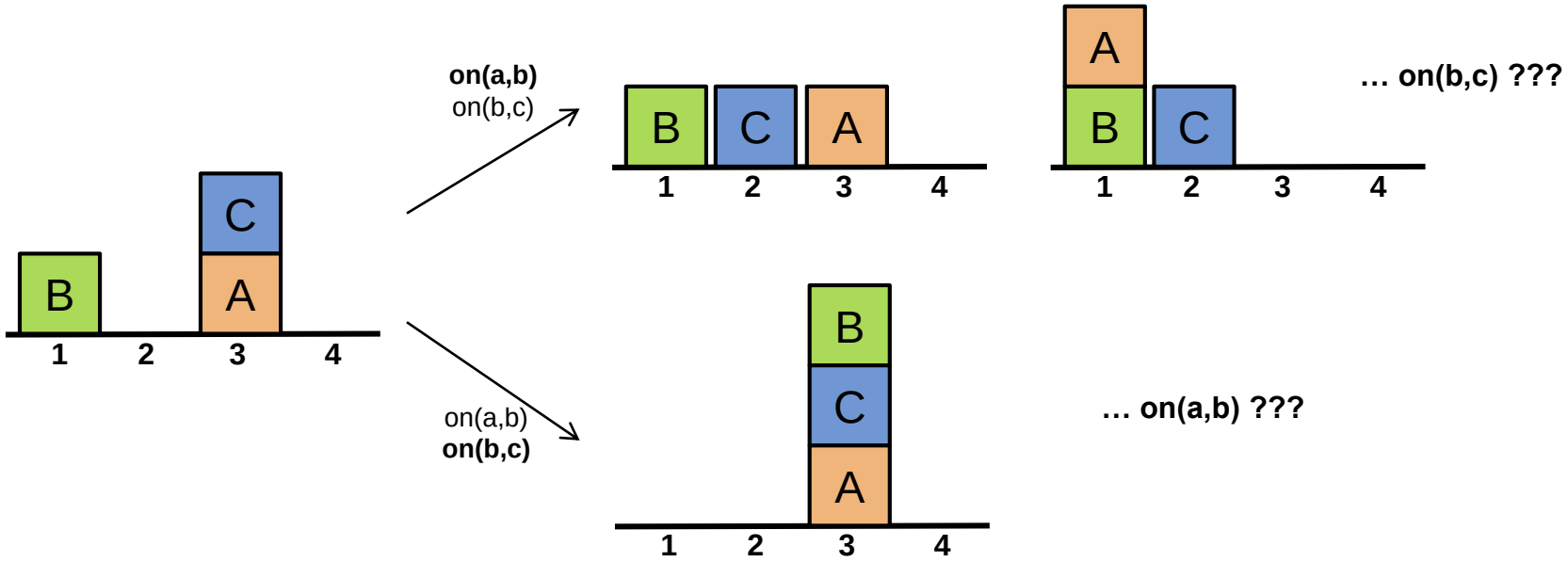
- pri iskanju v širino / iterativnem poglabljanju dobimo naslednjo rešitev:

```
move(c,a,2)   % doseže clear(a) za move(b,3,a)
move(b,3,a)   % doseže on(b,a) za move(b,a,c)                idealno bi bilo move(b,3,c) ?
move(b,a,c)   % doseže clear(a) za move(a,1,b) / doseže on(b,c)
move(a,1,b)   % doseže on(a,b), vsi cilji izpolnjeni
```

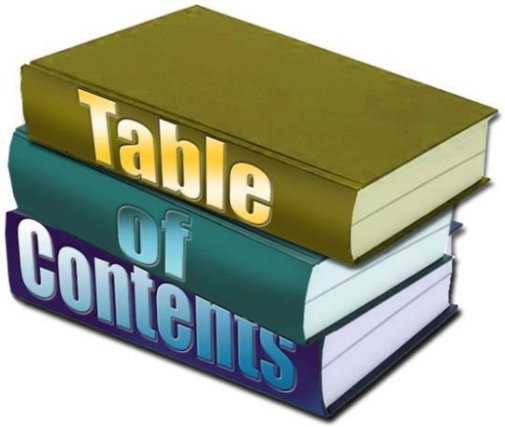
- najkrajši plan ni (vsebinsko) optimalen?
- zakaj?

Sussmanova anomalija

- Sussman anomaly (Gerald Sussman, 1975)
- je problem interakcije med cilji
  - algoritem za planiranje (STRIPS) obravnava cilje "lokalno" (enega po enega, ne ozirajoč se na drugega med reševanjem prvega)
  - z doseganjem enega cilja algoritem razveljavi že dosežene cilje ali predpogoje za njihovo doseganje
  - planiranje poteka linearno (najprej prvi cilj, šele nato naslednji, …)
  - vrstni red obravnavanja ciljev vpliva tudi na nepotrebne korake pri planiranju
- rešitve
  - drugačen algoritem za planiranje (regresiranje ciljev)
  - ne vztrajaj na urejenosti ciljev, če to ni nujno potrebno (nelinearno planiranje)



Pregled

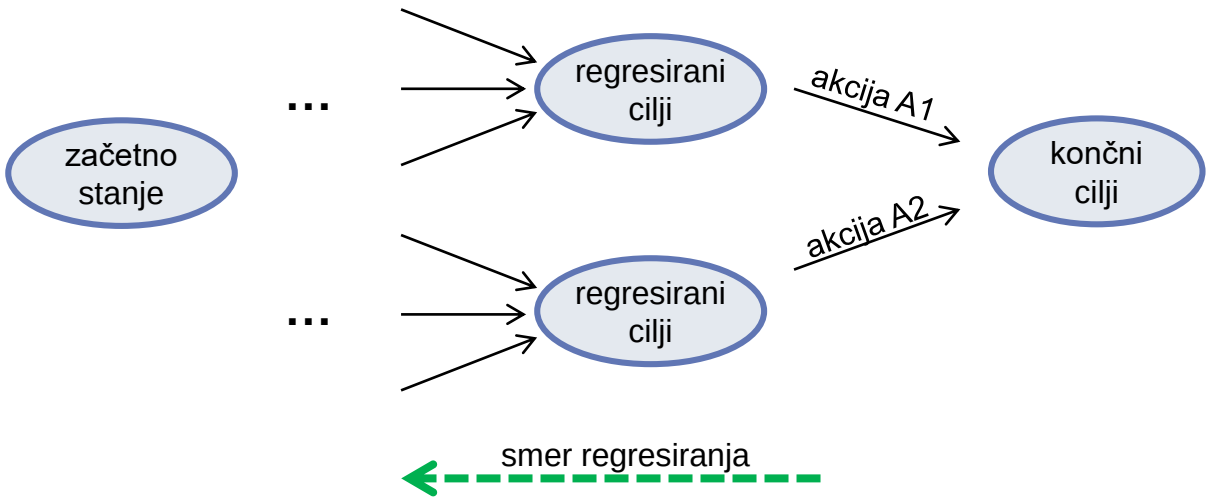


III. PLANIRANJE in razporejanje opravil

- predstavitev problema
- planiranje s "klasičnim" preiskovanjem prostora stanj
- planiranje s sredstvi in cilji
- planiranje z regresiranjem ciljev
- razporejanje opravil

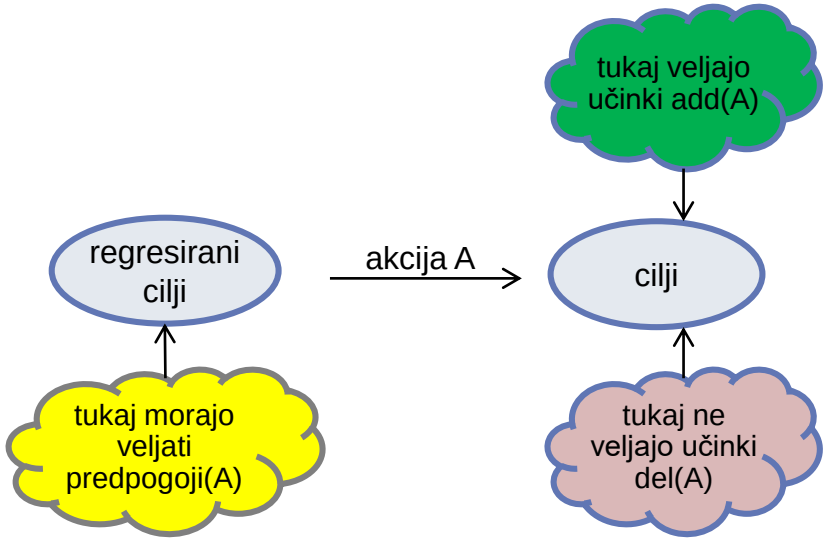
Planiranje z regresiranjem ciljev

- rešitev za Sussmanovo anomalijo
- vzvratno preiskovanje od cilja proti začetnemu stanju (angl. *goal regression through action*)
- drugačna filozofija:
  - *globalno* planiranje, ker algoritem za planiranje obravnava vse cilje hkrati
  - ne obravnavamo samo akcij, ki so ***možne***, temveč *najbolj smiselne*
- postopek:
  - izberemo akcijo, ki doseže izbrani cilj (in čim več preostalih ciljev)
  - izračunamo "predhodne" cilje ob uporabi te akcije (= regresiranje ciljev skozi akcijo)
  - nadaljujemo z regresiranjem, dokler ne pridemo do ciljev, ki so izpolnjeni v začetnem stanju



Planiranje z regresiranjem ciljev

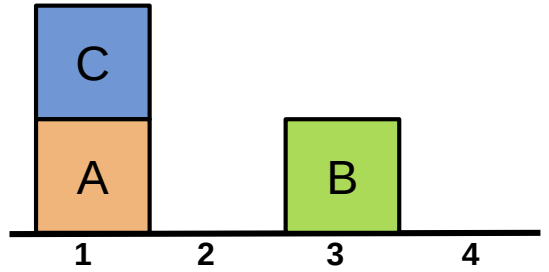
- postopek regresiranja ciljev



- $\text{regresirani cilji} = \text{cilji} \cup \text{predpogoji}(A) - \text{add}(A)$
- veljati mora $\text{cilji} \cap \text{del}(A) = \emptyset$

- "stanja" pri preiskovanju so **množice ciljev**
- **ciljni pogoj:** $\underline{\text{regresirani cilji} \subseteq \text{cilji}}$ v začetnem stanju
- uporabimo znane preiskovalne algoritme (neinformirani / informirani algoritmi; A*, hevristika?)

Planiranje z regresiranjem ciljev



- v primeru iz sveta kock velja:
  stanje: `[on(c,a), on(a,1), on(b,3), clear(c), clear(2), clear(b), clear(4)]`
  cilj: `[on(a,b),on(b,c)]`

- rešitev z regresiranjem ciljev najde optimalno rešitev

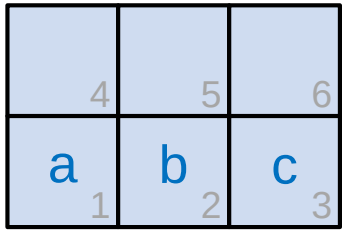
```
move(c,a,2)
move(b,3,c)
move(a,1,b)   %plan zaključen, vsi cilji izpolnjeni
```

- vaja regresiranja:
  Regresiraj cilja $C = \{on(a,b), on(b,c)\}$ skozi akcijo `move(a,2,b).`

$$Regresirani\ cilji\ = C \cup predpogoji(A) - add(A)$$
$$= \{on(a,b), on(b,c)\} \cup \{clear(a), clear(b), on(a,2)\} - \{on(a,b), clear(2)\} =$$
$$= \{on(b,c), clear(a), clear(b), on(a,2)\}$$

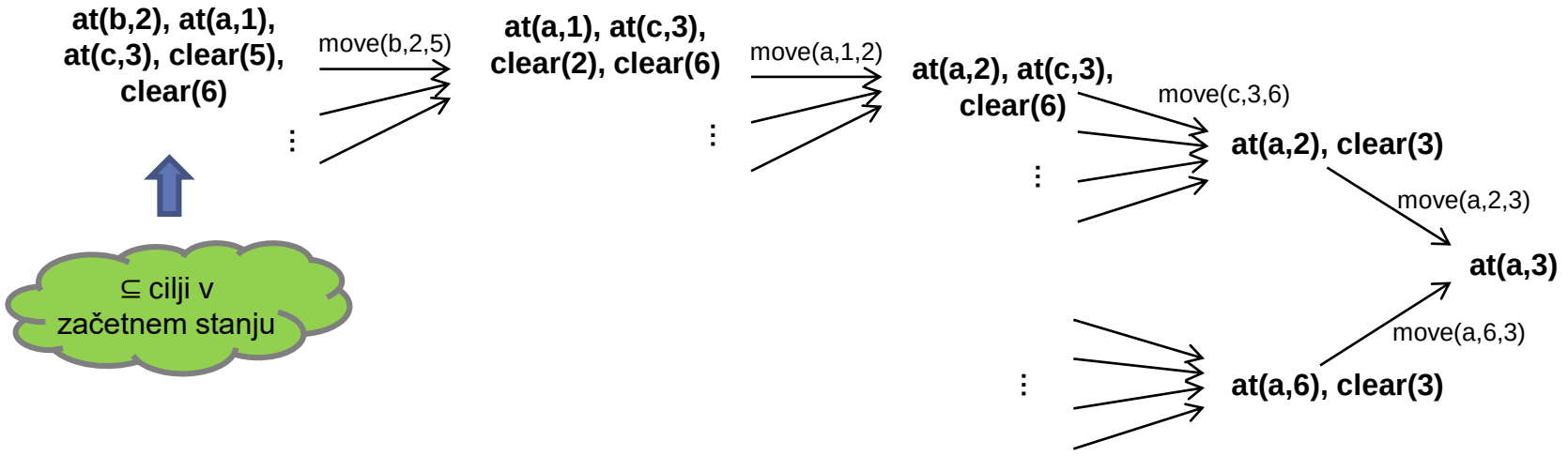
Pogoj: $\{on(a,b), on(b,c)\} \cap \{on(a,2), clear(b)\} = \emptyset$

Planiranje z regresiranjem ciljev



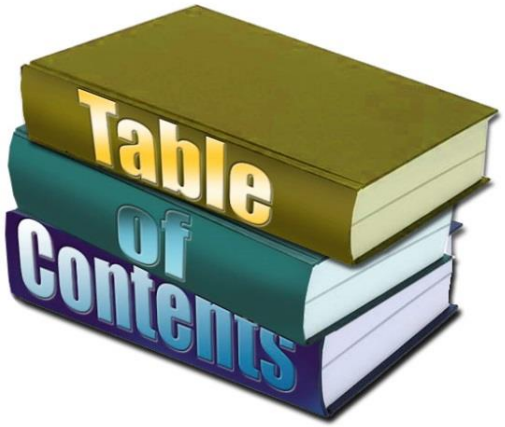
- primer: roboti na pravokotni mreži
- začetno stanje: `[at(a,1), at(b,2), at(c,3), clear(4), clear(5), clear(6)]`
- ciljno stanje: `[at(a,3)]`
- akcija:

	`move(Robot,From,To)`
predpogoj:	`[at(Robot,From), clear(To)]`
implicitne omejitve:	`[robot(Robot), adjacent(From,To)]`
add:	`[at(R,To), clear(From)]`
del:	`[at(Robot,From),clear(To)]`



- plan: `move(b,2,5), move(a,1,2), move(c,3,6), move(a,2,3)`

Pregled

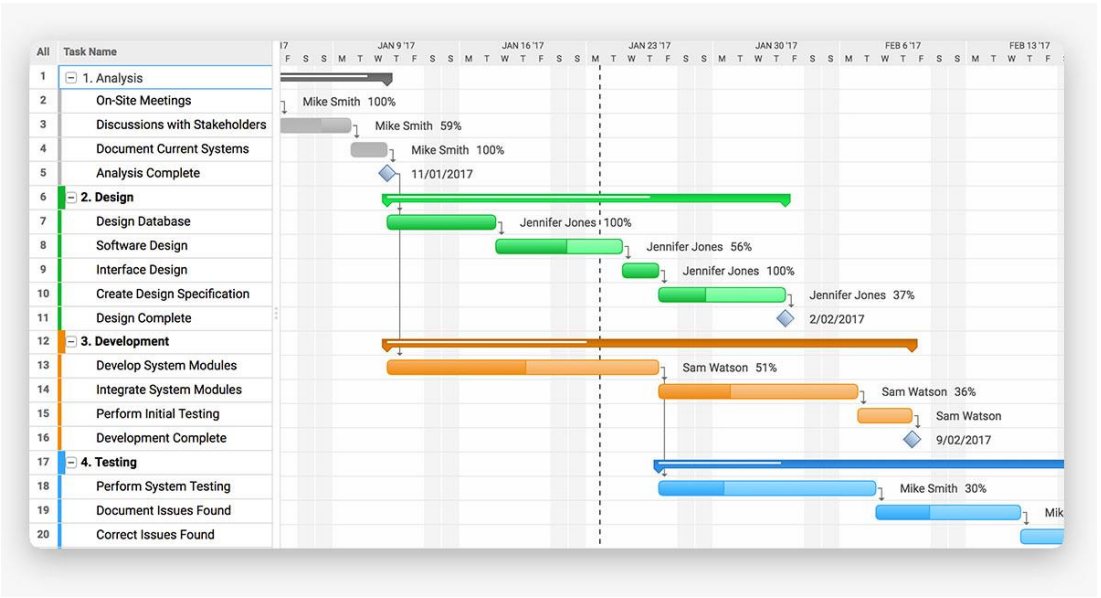


III. PLANIRANJE in razporejanje opravil

- predstavitev problema
- planiranje s "klasičnim" preiskovanjem prostora stanj
- planiranje s sredstvi in cilji
- planiranje z regresiranjem ciljev
- razporejanje opravil

Planiranje in razporejanje opravil

- do sedaj (klasično planiranje): **kaj narediti** in v kakšnem **vrstnem redu**
- pristopi:
  - planiranje kot preiskovanje prostora stanj
  - planiranje s sredstvi in cilji
  - planiranje z regresiranjem ciljev skozi akcije
- v realnosti imamo številne **dodatne omejitve**:
  - **časovne omejitve** (začetki aktivnosti, trajanja aktivnosti, roki zaključkov)
  - **resursi** (omejeno število procesorjev, kadra, bencina, denarja, surovin, …)



Razporejanje opravil

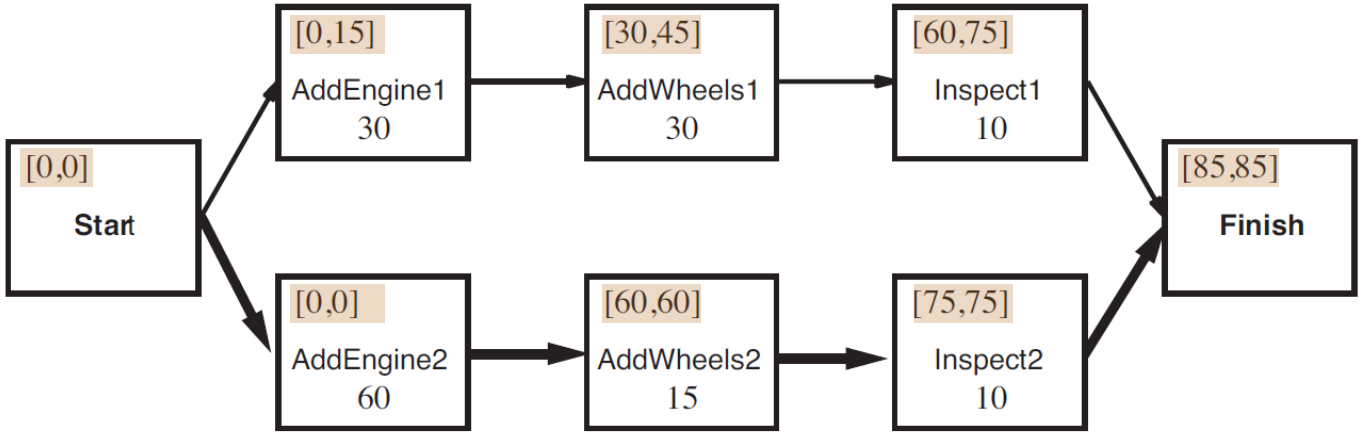
- delno urejen plan: vrstni red podmnožice aktivnosti je lahko urejen
- razširimo lahko notacijo (PDDL):
  - **`Akcija1 ≺ Akcija2`**: pomeni, da se mora Akcija1 zgoditi pred Akcijo2
  - **Resources** podaja števila razpoložljivih resursov
  - **DURATION** opredeljuje trajanje posamezne akcije
  - **CONSUME** opredeljuje (trajno) porabo določene količine resursov
  - **USE** opredeljuje (začasno) zasedenost količine resursov med izvajanjem akcije

```
Jobs (AddEngine1 ≺ AddWheels1 ≺ Inspect1,
      AddEngine2 ≺ AddWheels2 ≺ Inspect2 )
Resources (EngineHoists(1), WheelStations(1), Inspectors(2), LugNuts(500))

Action (AddEngine1 , DURATION:30,
        USE:EngineHoists(1))
Action (AddEngine2 , DURATION:60,
        USE:EngineHoists(1))
Action (AddWheels1 , DURATION:30,
        CONSUME:LugNuts(20), USE:WheelStations(1))
Action (AddWheels2 , DURATION:15,
        CONSUME:LugNuts(20), USE:WheelStations(1))
Action (Inspect i, DURATION:10,
        USE:Inspectors (1))
```

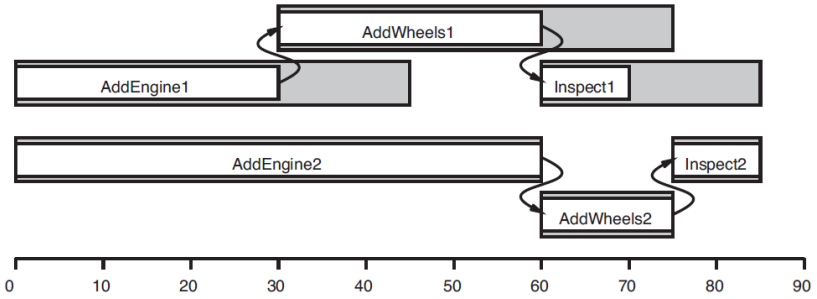
Razporejanje opravil

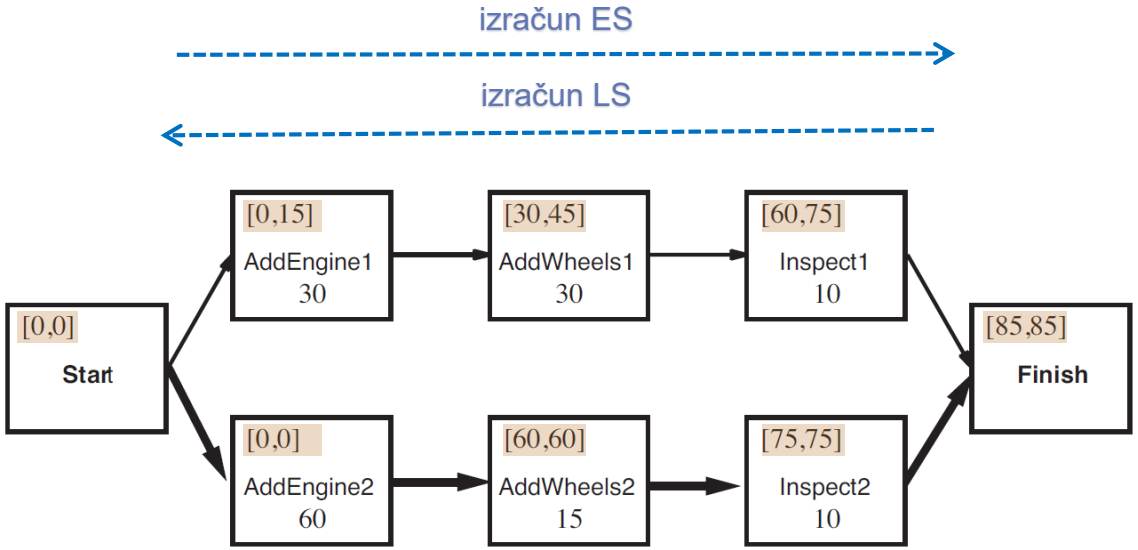
- za začetek: samo časovne omejitve
- **metoda kritične poti**
  - kritična pot: pot, ki je najdaljša in določa dolžino trajanja celotnega plana (krajšanje vzporednih poti ne vpliva na trajanje plana)
  - vsaki akciji priredimo par **[ES, LS]**:
    - **ES** – najbolj zgodnji možen začetek (angl. *Earliest Start*)
    - **LS** – najbolj pozen možen začetek (angl. *Latest Start*)



Razporejanje opravil

$$ES(Start) = 0$$
$$ES(B) = \max_{A \prec B}[ES(A) + Duration(A)]$$
$$LS(Finish) = ES(Finish)$$
$$LS(A) = \min_{A \prec B}[LS(B) - Duration(A)]$$
$$rezerva\ (slack) = LS - ES$$

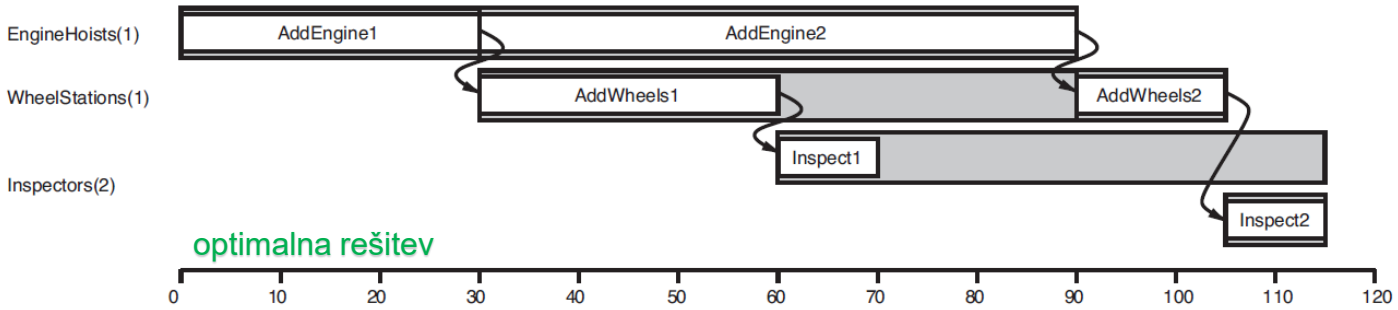




- časovna zahtevnost algoritma: $O(Nb)$, N – število akcij, b – faktor vejanja

Razporejanje opravil

- dodatno: upoštevanje tudi resursov
- uvede **omejitev**, da se aktivnosti, ki potrebujeta iste resurse, ne smeta prekrivati



- sprememba časovne zahtevnosti: $O(Nb)$ → NP-težek problem (!)
- primer izziv iz leta 1963 nerešen 23 let:
  - resursi: 10 strojev, 10 nalog, 100 akcij
  - preizkušene metode: simulirano ohlajanje, tabu search, razveji in omeji, …
- primerna hevristika: algoritem **najmanjše časovne rezerve** (angl. *minimum slack algorithm*)
  - na vsaki iteraciji dodeli **najbolj zgodnji možen začetek** akciji, ki ima **izpolnjene vse predhodnike** in ima **najmanj časovne rezerve**,
  - nato posodobi [ES in LS] za celotni graf in ponovi.

Razporejanje opravil

- Kakšen je rezultat simulacije algoritma najmanjše časovne rezerve na obravnavanem problemu?
- Ali je rešitev enaka optimalni? Zakaj?

- Kako upoštevati omejitve v zaporedju akcij pri pristopih za planiranje?
- Kako upoštevati omejitve v omejenem številu resursov?

Primer izpitne naloge

- 3. izpit, 2. 9. 2019

2. NALOGA (10t):

Podan je naslednji delno urejen plan s trajanji akcij, njihovimi odvisnostmi in uporabo resursov:

```
Jobs (Zajtrk<Kosilo<Vecerja, Kava<Caj)
Resources(Salica(1), Lonec(1))
Action (Zajtrk, DURATION:10, USE:Salica(1))
Action (Kosilo, DURATION:15)
Action (Vecerja, DURATION:10, USE:Lonec(1))
Action (Kava, DURATION:30, USE:Salica(1))
Action (Caj, DURATION:15, USE:Lonec(1))
```

a) **(7t) Na zgornjem planu simuliraj *algoritem najmanjše časovne rezerve* (angl. minimum slack algorithm) in z njim določi plan izvajanja (grafično).**

b) **(3t) Ali je algoritem v točki a) našel optimalno rešitev? Če ne, predlagaj boljšo (na pamet, brez simulacije algoritma).**