1. Upravte program na str. 316 tak, aby zisťoval, čí súčet niektorých prvkov je ostro medzi M1 a M2, M1 < M2. Pomocou safety a progress vlastností dokážte, že funguje, t.j. robí čo chceme.

Ak naše M1 a M2 sú kladne celé čísla tak musíme uvažovať také čísla M1 a M2 ktorých rozdiel spĺňa túto podmienku (M2 – M1) > 1. Pretože ak ich rozdiel by bolo číslo 1 tak pre takéto čísla nemá zmysel používať tento program pretože už rovno to nemá riešenie. Zoberme si napríklad M1 = 4 a M2 = 5 tak pri takýchto číslach to už rovno nemá riešenie pretože neexistuje číslo sum typu integer, ktoré by spĺňalo takúto podmienku 4 < sum < 5.

Program

```
declare sum, v : integer, seq : sequence of integer
always  sum = ⟨+ i: i ∈ seq :: A[i] ⟩ + A[v]
initially seq, v = null, 0
assign
seq, v := seq;v, v+1                    if sum <= M1

        ~ pop(seq), top(seq) + 1        if sum >= M2
end
```

Pridáme posledný prvok do A, A[N]=M2 - 1. Dávame tam najväčší možný prvok ktorý spĺňa naše zadanie a to je M2 - 1 pretože chceme byť ostro medzi M1 a M2 teda nám bude platiť M1 < M2-1 < M2. Tým vlastne zaručíme že úloha má riešenie pretože prinajhoršom tento samostatný prvok sám osebe spĺňa našu podmienku aby súčet bol ostro medzi M1 a M2. Samozrejme že hľadáme iné riešenie a keď žiadne iné riešenie nenájdeme teda budeme mať len toto tak to bude znamenať že tá úloha nemá riešenie. Snažíme sa zistiť či existuje taká postupnosť kde M1 < sum < M2 a neobsahuje N.

Safety vlastnosť:

Chceme ukázať že nič zlé sa nestane – postupnosť (seq;v) bude vždy rastúca, to znamená že nikdy počas výpočtu sa nám nestane že by sme rastúcu postupnosť pokazili a dali nakoniec tej postupnosti menší index ako je jeho ľavý sused a žiadna postupnosť lexikograficky menšia ako (seq;v) nebude riešením.

Naša safety vlastnosť je: (seq;v) je rastúca postupnosť indexov z 0..N a žiadna postupnosť lexikograficky menšia ako (seq;v) nie je riešením je celé *invariant*

Chceme vlastne ukázať že (seq;v) je stále rastúca postupnosť a žiadna postupnosť lexikograficky menšia ako (seq;v) nie je riešením. Ukážeme to pomocou Programu a jeho priradení.

- seq, v := seq;v, v+1 if sum <= M1
  ~ pop(seq), top(seq) + 1 if sum >= M2

Máme tu len toto jedno priradenie a to nám náš invariant nekazí, pretože ak platí v tomto priradení prvá podmienka tak do postupnosti indexov pridáme index ktorý je väčší ako ten posledný v tej postupnosti a zároveň zvýšime v o jedna pre nadchádzajúce priradenie kde toto v bude zasa väčšie ako posledný index v postupnosti takže nám to rastúcu postupnosť nekazí a keďže na začiatku inicializujeme v na nulu tak program začína vždy od najmenšieho indexu. Ak platí druhá podmienka v tomto priradení tak odstránime z konca postupnosti náš najväčší index čo nepokazí invariant a ešte k tomu do v priradíme index väčší než ten ktorý sme odstránili takže sa nám nemôže stať že by sme do v dali nejaký menší index ako ten ktorý sme odstránili. Takže od ďalšieho priradenie budeme keď tak pridávať do postupnosti ešte väčší index než ten ktorý sme odstránili, toto nám rastúcu postupnosť nepokazí. Týmto priradením ani nemôžeme preskočiť nejaký index že by sme ho vynechali v riešení pretože prvou podmienkou sa vždy posúvame o jeden index ďalej a druhou ak odstránime posledný index tak ako ďalšie v je index ktorý bol o jedna väčší ako ten odstránení takže ani tu nepreskakujem indexy, takže nám to ani nekazí invariant ktorý hovorí o postupnosti lexikograficky menšej ako (seq;v), ktorá nie je riešením.

Progress vlastnosť:

Chceme ukázať že sa urobí niečo dobre – nakoniec raz určite bude platiť že M1 < sum < M2.

Naša progress vlastnosť je: true *leads-to* M1 < sum and sum < M2

Chceme ukázať že ak bude platiť true tak niekedy raz bude platiť aj M1 < sum < M2, keďže true platí vždy tak určite bude platiť aj táto podmienka M1 < sum < M2. Ukážeme to pomocou priradenia a Programu.

- seq, v := seq;v, v+1 if sum <= M1
  ~ pop(seq), top(seq) + 1 if sum >= M2

Pomocou tohto priradenia a pomocou už dokázanej safety podmienky ktorá hovorí že postupnosť (seq;v) bude vždy rastúca a že nikdy nepreskočíme nejaké riešenie že by sme vynechali nejaký index, tak budeme postupne prechádzať všetkými rastúcimi postupnosťami to znamená že raz sa dostaneme nakoniec, môžu nastať 2 prípady:

1) narazíme na postupnosť (seq;v) ktorá neobsahuje index N a bude platiť M1 < sum < M2 to znamená že v priradení nebude platiť ani jedna podmienka a už sa priradenia nikdy kvôli tomu nebudú môcť vykonať.
2) Keďže sme do poľa A pridali nakoniec prvok A[N]=M2 - 1, tak ak prvý prípad nenájde nejakú postupnosť, ktorá by vyhovovala podmienke M1 < sum < M2 tak sa určite dostaneme na nakoniec a do stavu kde bude nejaká postupnosť

(seq;v) kde v = N a bude splnená podmienka M1 < sum < M2, pretože už len samotne v = N spĺňa túto podmienku pretože M1 < M2-1 < M2.

Ukázali sme že raz bude platiť naša podmienka M1 < sum a zároveň sum < M2.

Ukázali sme že pomocou safety, progress podmienky a nekonečného výpočtu vždy sa raz dostaneme do FP, to znamená že sa dostaneme do stavu kde každým vykonaním nášho jedného priradenia sa už nezmení postupnosť pretože podmienky v priradení nebudú pravdivé a teda nebude sa môcť to priradenie už vykonať. Tým pádom sme našli takú postupnosť (seq;v) ktorá spĺňa podmienku M1 < sum < M2. Ak v ≠ N pre FP tak pravé vtedy sme našli postupnosť (seq;v) ktorá je našim riešením.

2. Program na str. 322 upravte tak, aby ste nepoužívali always sekciu. Dokážete, že výsledok je skutočne minimum.

Program

initially ⟨▯ i: 1 ≤ i ≤ N :: g[i,i] = 0 ⟩
assign
⟨▯ i,j: 1 ≤ i < j ≤ N :: g[i,j] = ⟨min k: i ≤ k < j :: g[i,k] + g[k+1,j] + r[i-1] x r[k] x r[j] ⟩ ⟩

end

Dokážeme, že výsledok je skutočne minimum. Vidíme že v g máme pri rovnakých obidvoch indexoch danú hodnotu 0 a pri ostatných indexoch takých že i ≠ j pre g[i,j] máme nejakú zlú hodnotu pretože pri initially im žiadnu špecifickú hodnotu nedávame. Na začiatku pri jednotlivých priradeniach budeme pridávať do g minimá ktoré nebudú pravdivé pretože máme vo veľa prvkoch g zlé hodnoty ktoré nezodpovedajú tým správnym hodnotám ale keďže máme nekonečný výpočet tak postupnými priradeniami typu ako napr. pre g[2,3] ľahko dostaneme jeho minimum pretože jediné k môžeme zobrať k = 2 a teda pri g[2,2] mame 0, pri g[3,3] dostaneme 0 a potom nám stačí len vypočítať r[2-1] x r[2] x r[3] čo vôbec nezávisí na hodnotách ktoré by boli nepravdivé pretože tieto hodnoty sú dané dimenziou matíc a tie sa nemenia to znamená že hodnoty, ktoré by nám to mohli pokaziť sú tie z g ale keďže tie sú nulové tak nám to nekazia a takým to spôsobom získame minimum pre hodnoty g ktorých rozdiel ich indexov alebo inak povedane vzdialenosť medzi nimi sa rovná 1 a potom následne ľahko získame aj hodnoty pre g ktoré majú rozdiel indexov rovný 2 a tak ďalej. Takýmto postupným výpočtom po jednotlivých indexoch dostaneme všade správne minimum hodnoty pretože máme nekonečný výpočet a každé priradenie sa vykoná nekonečne veľa krát.

3. Program na str. 330. Pomocou safety a progress vlastností dokážte, že funguje, t.j. robí čo chceme.

Safety vlastnosť:

Chceme ukázať že nič zlé sa nestane – ak sa dostaneme do dest to znamená že u = dest tak sa nemôže stať že by sme tento vrchol dest opustili, že by sme sa vybrali na iný vrchol.

Naša safety vlastnosť je: u = dest *is stable*

Chceme ukázať že ak raz sa stane u = dest tak už nikdy tento vrchol neopustíme, stále potom bude platiť že u = dest. Keďže máme nekonečný výpočet tak vlastne jednotlivými priradeniami v Programe sa nám stane že u = dest a teraz to ideme preskúmať po jednotlivých priradeniach, že či dokážu zmeniť náš vrchol u z dest na nejaký iný.

- u,p[v] := v,u if v ≠ N ∧ u ≠ dest {rozšírenie cesty}

Toto priradenie nám neškodí pretože nás nemôže dostať z vrcholu dest na nejaký iný vrchol. Pretože ak už raz platí že u = dest tak toto priradenie sa nevykoná pretože podmienka bude false, lebo máme u = dest a to porušuje danú podmienku.

- u,p[u] := p[u], N if v = N ∧ u ≠ 0 ∧ u ≠ dest {backtrack}

Toto priradenie nám tiež nepokazí náš stav, že by sme sa z dest dostali do nejakého iného vrcholu, pretože ak už raz platí u = dest tak podmienka v tomto priradení nikdy nebude true tak tým pádom nám toto priradenie neškodí.

Progress vlastnosť:

Chceme ukázať že sa urobí niečo dobre – ak sa nachádzame vo vrchole u = 0 tak raz sa určite dostaneme do u = dest, čiže nám to hovorí že nájdeme cestu z 0 do dest.

Naša progress vlastnosť je: u = 0 *leads-to* u = dest

Chceme ukázať že ak sa nachádzame vo vrchole u = 0 a keďže to je vždy začiatočný vrchol tak to platí, tak sa raz dostaneme do u = dest, nájdeme takú cestu. Ukážeme to hlavne pomocou always sekcie a aj pomocou vrcholu N ku ktorému sme pri initially dali hranu ku každému vrcholu v grafe v Programe. Môžu nastať dva prípady a ten prvý je že začiatočný vrchol u je aj tým vrcholom dest, tak to nám rovno platí, my sa teraz ideme venovať druhému prípadu a to je keď máme viac vrcholov a vrchol dest nie je vrchol u = 0.

- always
  v = ⟨min j: 0 ≤ j ≤ N ∧ E[u,j] ∧ p[j] < 0 :: j ⟩

Tato always sekcia je pre nás dôležitá lebo vlastne hovorí že prejdeme cez celý strom, že nevynecháme žiadny vrchol cez ktorý by mohla viesť cesta. Vrchol v k vrcholu u vyberáme že je najmenším indexom a susedom s vrcholom u s ktorým je spojený hranou a zároveň ten vrchol v ešte nebol navštívený to znamená že sa dostaneme do vetvy vrcholu v a ak tam nič nie je že sme nenašli dest tak sa potom vrátime naspäť a znova sa vyberá vrchol v lenže ten predchádzajúci v z ktorého sme sa vrátili už v možnosti nebude pretože sme si ho označili pomocou priradenia {backtrack} že tam nič nie je, teda už bol navštívený tak vyberáme ďalšie minimum lenže toto minimum už bude väčší index ako ten predchádzajúci z ktorého sme sa

vrátili. Tým pádom ako keby prechádzame všetkých susedov vrcholu u s ktorými sme spojený hranou a ktorých sme ešte nenavštívili a mohli by byť potenciálne v našej ceste pretože taký ktorý sú už v našej ceste a taký v ktorých cesta byť určite nemôže nás nezaujímajú, teda prechádzame od najmenších indexov vrcholov v až po najväčšie ktoré spĺňajú tie podmienky v always sekcii, to znamená že žiadny vrchol nepreskočíme. Ak sa dostaneme do situácie že žiadny taký vrchol v už neexistuje že nemáme žiadny vrchol v ako suseda ktorý by nebol navštívený tak vždy tam zostane vrchol N ktorý je náš záchranný vrchol v situácii keď už nemáme kam ísť a pomocou tohto vrcholu sa zo zlej vetvy kde nie je dest dostaneme do vrcholu nášho predchodcu a následne pomocou priradenia {backtrack} si označíme že do tejto vetvy už nikdy nechoďme. Vrchol N je dôležitý pretože pomocou neho sa dostaneme zo zlých vetiev v ktorých by sme sa inak zasekli. Takýmto spôsobovom prejdeme cez všetky možne cesty grafu a nájdeme nakoniec cestu k vrcholu dest a dostaneme teda u = dest. Ak sa po ceste zasekneme tak nám vždy pomôže vrchol N pretože k tomuto vrcholu vedu hrany zo všetkých ostatných vrcholov v grafe.

Ukázali sme že pomocou safety a hlavne progress podmienky a nekonečného výpočtu vždy sa raz dostaneme do FP, to znamená že sa dostaneme do stavu kde každým vykonaním priradení sa nám nič nezmení pretože podmienky v priradeniach budú stále false a teda nebudú sa môcť tie priradenia vykonať. Tým pádom sme sa dostali do u = dest a našli cestu z vrcholu 0, čo bolo aj to čo sme chceli.