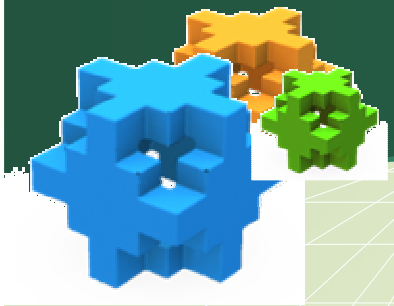
Dinamičko programiranje 2

Goran Žužić <zuza777@gmail.com>



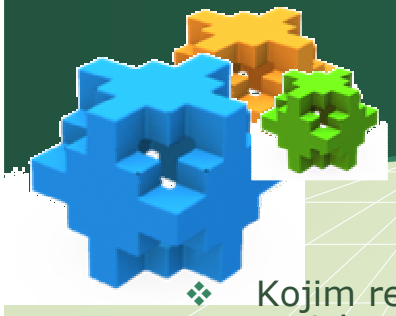
Kratki sadržaj

Zadaća - Piramida

- Paradigma: Dinamičko programiranje!

5	x	x	…
2	4	x	…
7	9	2	…
…	…	…	…

- Osnovna ideja: na slici desno, je li moguće da put označen crvenom bojom ikada bude dio optimalnog rješenja? Zašto?
- Dovoljno je za svako polje pamtiti najbolji put (s najvećom sumom)
  - Gradimo matricu ***dp[x][y]***, neka nam je zadana ***A[][]***
  - ***dp[x][y] = max(dp[x-1][y], dp[x-1][y-1]) + A[x][y]***
    - Implementacijska zamka: paziti da polja kojima pristupamo postoje
  - Inicijalizacija: ***dp[0][0] = A[0][0]*** (...očito)
  - Rješenje: ***max{ dp[n-1][i] za 0 <= i < n }***
- Složenost:
  - Broj bitnih stanja u dinamici: ***1+2+…+n = n(n+1)/2***
  - Vrijeme potrebno za izračunavanje jednog stanja = 1 (konstantno vrijeme, optimalno)
  - Broj operacija: |Stanja|*SloženostPrijelaza ~ n(n+1)/2 <= 125 250 operacija za maksimalni ***n***
    - Današnja računala obavljaju otprilike $5*10^8$ jednostavnih operacija (zbrajanje, množenje, ali dijeljenje NE!) po sekundi
  - Memorijska složenost: n^2 (0.25 MB za max ***n***)

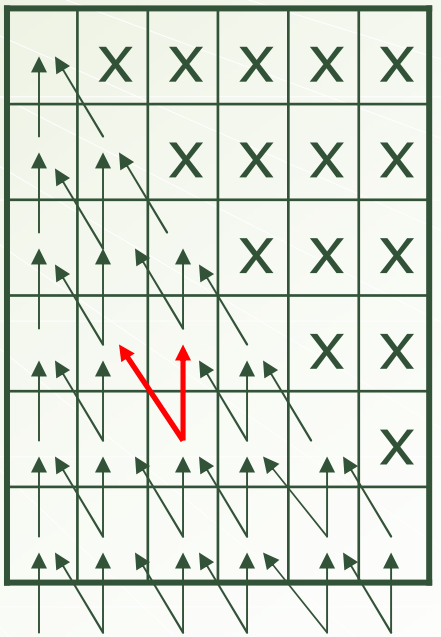


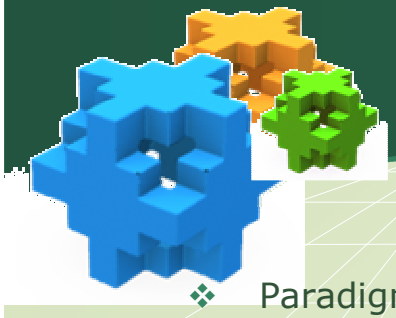
Piramida (nastavak)

- Kojim redoslijedom ćemo izračunavati vrijednosti matrice ***dp***?
  - Prije nego što pokušamo izračunati ***dp[x][y]*** trebamo biti sigurni da su ***dp[x-1][y]*** i ***dp[x-1][y-1]*** izračunati (pod uvjetom da postoje!).

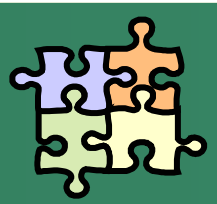
- Na slici desno vidimo međusobne ovisnosti (**preduvjete**). Nakon kratkog razmišljanja jasno je da možemo prvo lijeva na desno izračunati vrijednosti matrice ***dp*** u prvom retku, zatim opet slijeva nadesno u drugom retku, i tako dalje dok ne dođemo do ***n***-tog retka.
  - Svi preduvjeti se nalaze u prethodnom retku pa je jasno da su svi izračunati.
    - Matematički čistunci mogu slobodno navedeno svojstvo dokazivati indukcijom, unatoč tome što je točnost evidentna

- Čemu sve ovo? Većina toga rečenog na prethodna dva slidea vam nije novo?
  - Ideja je da postanete svjesni da kodiranje dinamike ima puno aspekata. U ovom (jednostavnom) zadatku većina aspekata je trivijalna, ali zato u nadolazećim zadacima neće biti.





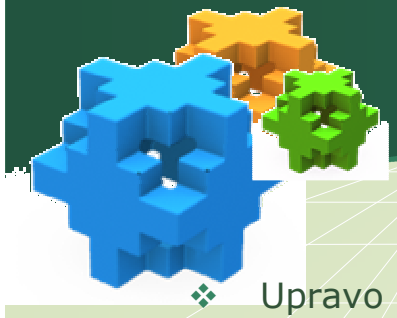
Zadaća - Podjela



- Paradigma: Dinamičko programiranje!

- Ekvivalentan zadatak: Na koliko načina možemo odabrati jednu grupu brojeva čija je suma jednaka točno polovici ukupne sume. Nakon toga samo podijelimo rješenje s 2 (zašto?).
  - Nije dobro! Budući da radimo s aritmetikom pod parnim modulom, ne znamo dijeliti s 2. Zahtijevali bismo aritmetiku proizvoljne preciznosti. (U Javi nam je to automatski dostupno, dok bi se u C/C++ morali malo pomučiti)
  - Napravit ćemo malu promjenu na zadatku: Na koliko načina možemo odabrati grupu brojeva čija je suma jednaka polovici ukupne sume, s time da ona obavezno sadrži prvi zadani broj? (dijeljenje nepotrebno ☺)

- Ideja: Pokušajmo generirati sve naše grupe jednu po jednu. Recimo da smo u izvjesnom trenutku razmotrili prvih 7 zadanih elemenata i odlučili smo s kojim podskupom želimo pokušati izgraditi jedno rješenje. Recimo da smo s njima dobili sumu 142. Iz informacija iz prethodne dvije rečenice vi biste trebali znati naći SVE načine na koji ja mogu dovršiti moju polu-izgeneriranu grupu tako da zadovoljava uvjete zadatka.

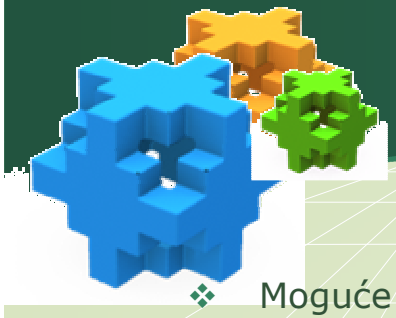


Podjela (nastavak)

- Upravo zato možemo riješiti zadatak dinamikom. Nema nam ***dp[k][s]*** označava na koliko načina možemo između prvih ***k*** elemenata odabrati grupu koja sadrži prvi zadani element, a njena suma je upravo ***s***. Neka nam je zadani niz (A).
  - ***dp[k][s] = dp[k-1][s] + dp[k-1][s-A[k-1]]***
    - Možemo odabrati da ne uzmemo ***(k-1)***. element (u tom slučaju iz prethodnih brojeva tražimo grupu sa sumom ***s***), ili u drugu ruku možemo uzeti ***(k-1)***. element gdje tražimo grupu sa sumom ***s-A[k-1]***

- Implementacija i aspekti dinamike:
  - Inicijalizacija: ***dp[?][?] = 0***, osim ***dp[1][A[0]] = 1***
  - Rješenje: ***dp[n][suma{A}/2]*** ako je suma parna, inače 0
  - ***dp[k][s]*** -> ***0<k<=n, 0<=s<=200n***
  - Složenost: ***|Stanja|*Prijelaz = 200n * 1*** <= 20 000
  - Redoslijed izračunavanja (jednostavan):

```
for( int k = 2; k < n; ++k )
        for( int s = 0; s <= 200*n; ++s ) {
            ...
```

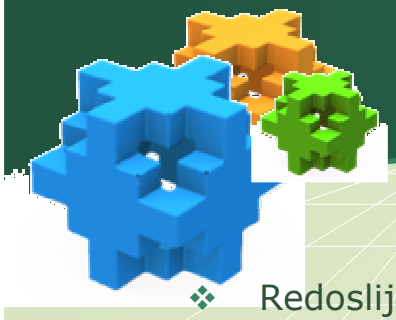


Zadaća - Palin

- Moguće rješenje: odaberimo sredinu našeg palindroma. Širimo se u obje strane dok god možemo. Broj operacija: n^2
  - Oprez: sredina palindroma može pasti i između dva slova (kod palindroma parne duljine)

- Moguće riješiti i pomoću dinamike. Neka je ***A*** oznaka za ulazni niz. Izračunat ćemo matricu ***dp*** takvu da ***dp[l][r] = 1*** ukoliko je ***A[l], A[l+1], ..., A[r]*** palindrom, inače je 0.
  - Nakon što je to izračunato, jednostavno možemo u n^2 operacija naći najdulji palindrom.
  - Relacija je jednostavna: ***dp[l][r] = dp[l+1][r-1] AND (A[l]==A[r])***

- U inicijalizacija samo ručno izračunamo vrijednosti matrice za sve podriječi duljine 1 i 2, to je jednostavno. Cijelo rješenje se može izvesti u otprilike n^2 operacija.



Palin (nastavak)

- Redoslijed izračunavanja dinamičke matrice više nije trivijalan. Slika će to ilustrirati.

(1,1)	(1,2)	(1,3)	(1,4)	(1,5)	(1,6)
X	(2,2)	(2,3)	(2,4)	(2,5)	(2,6)
X	X	(3,3)	(3,4)	(3,5)	(3,6)
X	X	X	(4,4)	(4,5)	(4,6)
X	X	X	X	(5,5)	(5,6)
X	X	X	X	X	(6,6)

podriječi duljine 6

podriječi duljine 5

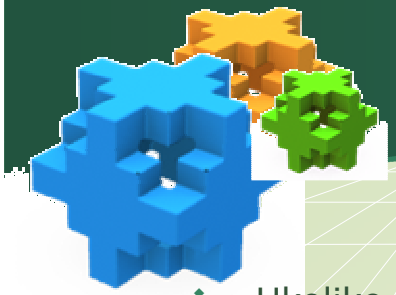
podriječi duljine 4 (pokazuju na preduvjete)

podriječi duljine 3

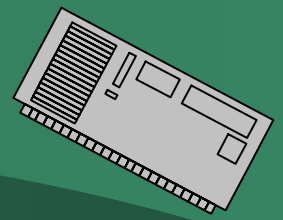
podriječi duljine 2

podriječi duljine 1

- Za razliku od prethodnih zadataka, tu ne smijemo izračunavati po *row-major* poretku. Ali vrlo mala promjena će biti dostatna. Uočimo da se svaka dijagonala izračunava iz dijagonale koja je ispod nje => Računajmo dijagonalu po dijagonalu! To je ekvivalentno analiziranju podriječi sortiranih po svojim duljinama.

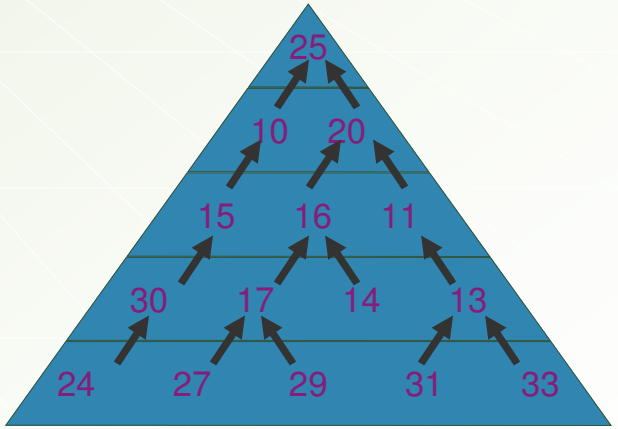


Dinamika i memorija

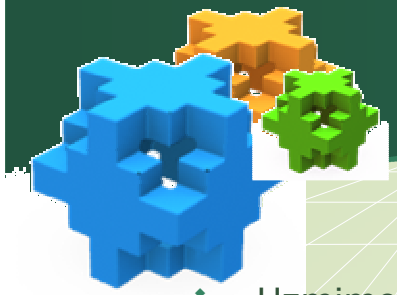


- Ukoliko ne želimo naći samo iznos najboljeg rješenja, nego želimo naći i njegovu "instancu", potrebna nam je dodatna memorija.
  - Ilustrirajmo to na poznatom primjeru: Ukoliko bi zadatak Piramida tražio ne samo da nađemo sumu na najboljem putu, nego da nađemo i *neki* najbolji put (tzv. *rekonstrukcija*), trebali bismo održavati i matricu ***prethodni[x][y]*** koja bi nam govorila s kojeg polja smo skočili na ***(x,y)*** u nekom najboljem rješenju kojeg smo pronašli do tog polja. Nakon toga je najbolji put lagano ispisati (kako?).

- Modno osviještena piramida (zimske boje):
  - Za razliku od one poravnate s lijevim rubom
  - Stvar prikaza, zadatak je isti
  - Prisjetimo se: Ako zamislimo brojke kao vrhove, a strelice kao neusmjerene bridove, kako nazivamo takav graf?



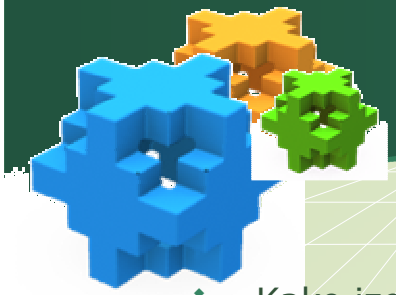
- Istina je da ne moramo uopće pamtiti matricu ***prethodni*** zato jer ***prethodni[x][y] = (x-1,y)*** ako i samo ako ***dp[x][y] = dp[x-1][y] + A[x][y]***
  - Neovisno o tome, ipak će nam u neku ruku trebati više memorije ukoliko trebamo rekonstruirati rješenje



Dinamika i memorija (nastavak)

- Uzmimo sada da ne moramo rekonstruirati. Originalno nam je trebalo otprilike $2n^2$ memorijskih riječi da bi spremili matrice ***A*** i ***dp***. Možemo li navedenu brojku drastično smanjiti?
  - Prisjetimo se relacije... ***dp[x][y] = max(dp[x-1][y], dp[x-1][y-1])***
  - Iz nje se jasno vidi da nam je za izračunavanje ***x***. retka matrice ***dp*** potrebno pamtiti ***(x-1)***. redak matrice ***dp*** te ***x***. redak matrice ***A***. U stvari, shvatimo da nam je u svakom trenutku dovoljno pamtiti samo prethodni redak ***dp*** (zaboravljamo one prethodne) i trenutni redak ***A*** (učitavamo redak po redak i zaboravljamo prethodne)
  - Metoda **rotirajućih matrica** *(retci se naizmjenično pune/prazne => rotiraju se)*
  - Memorijska složenost pada na ***3n*** (2 retka matrice ***dp*** i jedan redak matrice ***A***)!
    - Pretpostavimo da nam je u zadatku Piramida ograničenje na ***n*** bilo 10 000. Što je vjerojatnije? Da su nam na raspolaganju 2 sekunde i 32 MB ili da su nam na raspolaganju 1 sekunda i 512 MB? Situacija postaje uvjerljivija kada nabrijemo ***n*** na još veće vrijednosti.

- Memorijska poboljšanja na ostalim zadacima iz zadaće:
  - Palin (prethodna mem. složenost je $\sim n^2$): u relaciji nas zanimaju samo prethodne dvije dijagonale... Zašto bi pamtili išta ostalo (tu nas čak i rekonstrukcija ne smeta!)? Memorijska složenost pada na ***3n***.
  - Podjela (prethodna mem. složenost $\sim 200n^2$): Dovoljno je pamtiti samo podatke za prethodni broj. Memorijska složenost pada na ***200n***. Moguće je dovesti je i do ~***n***.



Dinamika i memorija (nastavak)

- Kako izgleda (elegantna) implementacija zaboravljanja prethodnih redaka (rotirajuće matrice)? Pogledajmo isječak iz koda zadatka Piramida:

```
dp[0][0] = A[0][0]; // podaci u matricu A su učitani
  for( int i = 1; i < n; ++i )
     for( int j = 0; j <= i; ++j ) {
        dp[i][j] = -inf;

        if( j-1 >= 0 && dp[i-1][j-1] > dp[i][j] ) dp[i][j] = dp[i-1][j-1];
        if( j <= i-1 && dp[i-1][j] > dp[i][j] ) dp[i][j] = dp[i-1][j];

        dp[i][j] += A[i][j];
     }
```

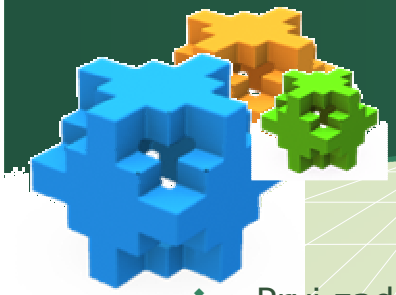
```
for( int i = 1; i < n; ++i ) {  // varijable n i dp[0][0] su učitani
      for( int j = 0; j <= i; ++j )
         scanf( "%d", &A[j] );

      for( int j = 0; j <= i; ++j ) {
         dp[i%2][j] = -inf;

         if( j-1 >= 0 && dp[(i-1)%2][j-1] > dp[i%2][j] ) dp[i%2][j] = dp[(i-1)%2][j-1];
         if( j <= i-1 && dp[(i-1)%2][j] > dp[i%2][j] ) dp[i%2][j] = dp[(i-1)%2][j];

         dp[i%2][j] += A[j];
      }
   }
```

- Ako nam je rekonstrukcija bitna, rijetko ćemo moći dolaziti do tako drastičnih memorijskih optimizacija. Doduše, postoje (napredne) specijalizirane metode kojima je moguće poboljšati memoriju na račun vremena izvršavanja.
  - Za ilustraciju, recimo da nam program radi $14n^2$ operacija i koristi $3n^2$ memorije. Moguće ga je "poboljšati" tako da radi $107n^2$ operacija i koristi tek $3n^{1.5}$ memorije, ali omogućava rekonstrukciju.
    - Implementacijska noćna mora!
    - CEOI 2005 (regionalna informatička olimpijada) – zadatak "Mobile Services" – samo jedan natjecatelj je uspio riješiti zadatak

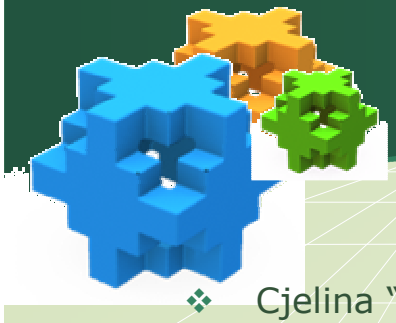


Radna pauza

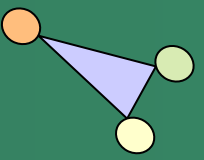
- Prvi zadatak: Snow Board
  - Mirko ide u Austriju isprobati svoj novi snowboard. Pred njim je planina reprezentirana s matricom dimenzija ***R***x***S*** gdje na mjestu ***(x,y)*** piše visina tog predjela. Mirko može jednokratno (helikopterom) doći na bilo koji dio planine i s jednog se kvadratića spušta samo na onaj koji je niže visine i ima barem jednu zajedničku stranicu. Koji je najduži put kojim se Mirko može spuštati?
    - Primjer i optimalno rješenje:
      - Odgovor: 7

8 ←	9	0	0
↓	↑		
7	10	0	99
↓		↑	
6 →	5 →	1	0

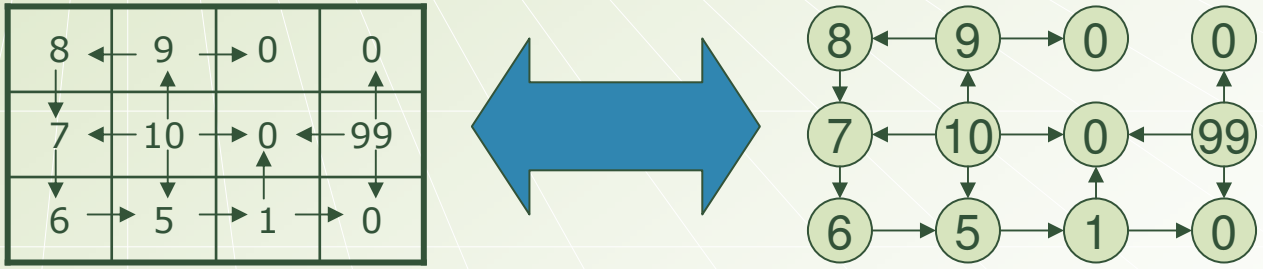
- Drugi zadatak: Brokule
  - Zadan je jednodimenzionalni niz ***(A)*** od ***n*** cijelih brojeva. Nasuprot tome, Slavko voli brokule i jede ih svaki dan. Zna se da je Slavko prvi dan pojeo ***A[0]*** brokula, a nakon toga se strogo držao sljedećeg pravila: u ***x***. danu je pogledao ***D*** dana unatrag i vidio kada je pojeo najmanje brokula, nazovimo tu najmanju količinu broj ***m***. Zatim bi pojeo ***A[x]+m*** brokula. Ritual je ponavljao sve do ***n***. dana. Odredite koliko je brokula Slavko pojeo na koje dane.



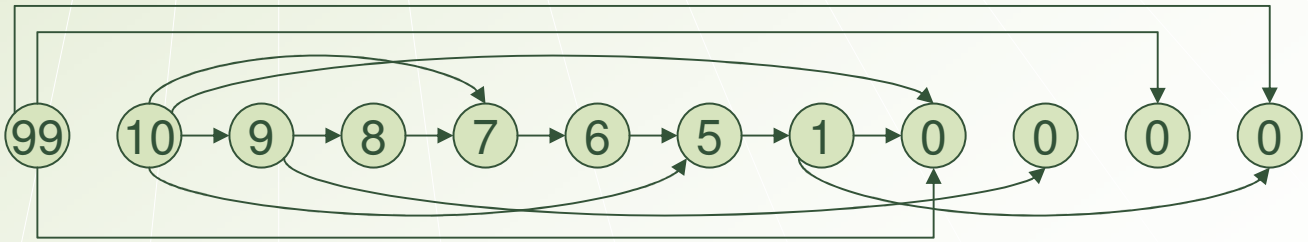
Dinamika i grafovi



- Cjelina "dinamika i grafovi" se bavi isključivo redoslijedom izračunavanja dinamičkih tablica. Zadatak Snow Board će biti glavni primjer takvog zadatka na kojem ću objasniti koncepte.
  - Stanje dinamike je očito: neka mi ***dp[x][y]*** označava maksimalnu duljinu spusta koji mogu napraviti ako započnem na kvadratiću ***(x,y)***
  - Ni relacija nije puno teža: ***dp[x][y] = 1 + max{ dp[x+1][y], dp[x][y+1], dp[x-1][y], dp[x][y-1] }***
  - Ali u kojem poretku da izračunavamo vrijednosti. Pogledajmo preduvjete prethodno danog primjera:



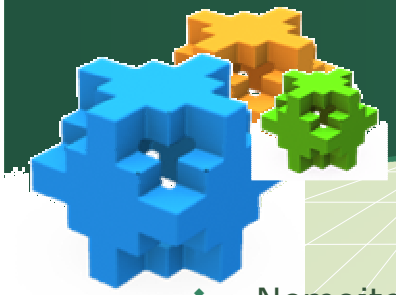
  - Ili kad malo preuredimo gornji graf:



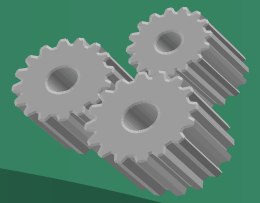
Dinamika i grafovi (nastavak)

- Primijetimo da svi bridovi udesno, tj. vrh iz kojeg brid kreće nikada nije desno od vrha u kojem brid završava. Takvo preuređivanje grafa se zove ***topološko sortiranje***. Treba imati na umu da uređaj prikazan na prethodnom slideu nije jedino moguće topološko sortiranje (npr. vrh 99 može noći bilo gdje, pod uvjetom da su sve 0 desno od njega).
  - Nakon što smo topološki sortirali vrhove, jasno nam je u kojem poretku ih trebamo obraditi. Prvo ćemo izračunati ***dp*** vrijednosti najdesnijih vrhova te ćemo se kretati sve ljevije i ljevije. Kako svi bridovi idu udesno, garantirano je da prije računanja nekog stanja svi preduvjeti biti ispunjeni.
  - U ovom zadatku se jasno vidi da će raditi i sljedeća metoda:
    - Prvo obradimo vrhove s najmanjom visinom i krećemo se postepeno prema višim
    - To znači da je sort kvadratića po visini ujedno i topološki sort (kao što je prikazano prethodnim grafom)
  - Sada razmislite o prethodnim zadacima i uvjerit ćete se da smo kod svih obrađivali tablicu dinamike u topološkom poretku

- Postoji li topološki sort uvijek? Odgovor je ne! Topološki sort postoji onda i samo onda ako na grafu nema ciklusa! Zato takve grafove zovemo **DAG**-ovima (**D**irected **A**cyclic **G**raph). Kada želimo riješiti problem dinamikom trebali bismo razmisliti je li pripadajući graf dinamike DAG ili ne.
  - Jedna mogućnost: Super, DAG je! Ako ne znate naći topološki sort automatski (npr. for petljama) upotrijebite svoj omiljeni algoritam za obilazak grafa (DFS, BFS?). Pazite da ništa ne računate 2 puta!
  - Druga mogućnost: Postoji ciklus ☹! Ovo je mnogo složenija mogućnost. Ponekad se još možete spasiti Dijkstrom ili Floydom, ali imajte na umu kako je moguće da rješenje ne postoji (npr. postoji negativni ciklus). Ali o tome jedan drugi put.



Dinamika i strukture

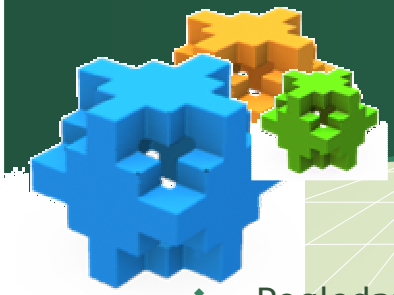


- Nemojte se obeshrabriti prethodnom cjelinom. U stvarnosti su zadaci puno lakši nego što vam se moglo učiniti gledajući prethodni slide.

- Dinamike sa strukturama su najvažnije natjecateljsko (i praktično) područje koje se veže za dinamičko programiranje (poslije osnovne ideje, naravno).

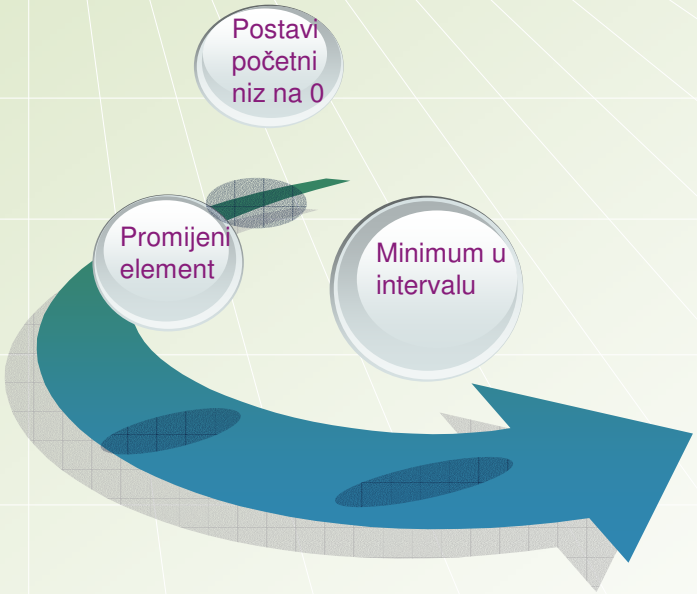
- Svakodnevna situacija: Imate dinamiku s relativno puno stanja i za svako od tih stanja trebate relativno puno operacija da biste odredili vrijednost (nazovimo tu drugu veličinu **složenost relacije**). Po poznatoj formuli ***puno*** x ***puno = previše***. Ideja koju ova cjelina nudi je jednostavna: smanjimo složenost relacije tako što ćemo upotrijebiti pametno osmišljenu strukturu podataka (o kojima ste učili u 3. tjednu predavanja ☺).

- Analizirajmo zadatak Brokule: Nije se teško sjetiti dinamike sa stanjem ***dp[x]*** = broj brokula koje je Slavko pojeo na ***x***. dan. Zadatak traži od nas da ispišemo cijeli taj niz. Inicijalizacija (***dp[0]=A[0]***) i poredak izračunavanja (***dp[0], dp[1], ..., dp[x-1]***) su trivijalni, a ni relacija nam ne predstavlja problem:
  - ***dp[x] = max{ dp[x-D], dp[x-D+1], ..., dp[x-1] } + A[x]***
  - Unatoč tome što je relacija relativno lagana, njena složenost je otprilike ***D*** operacija po stanju, što uz ***n*** stanja govori da je ukupna složenost rješenja ***n*D***. Previše, ukoliko se želimo igrati s vrijednostima tih varijabli do 10^5.



Dinamika i strukture (nastavak)

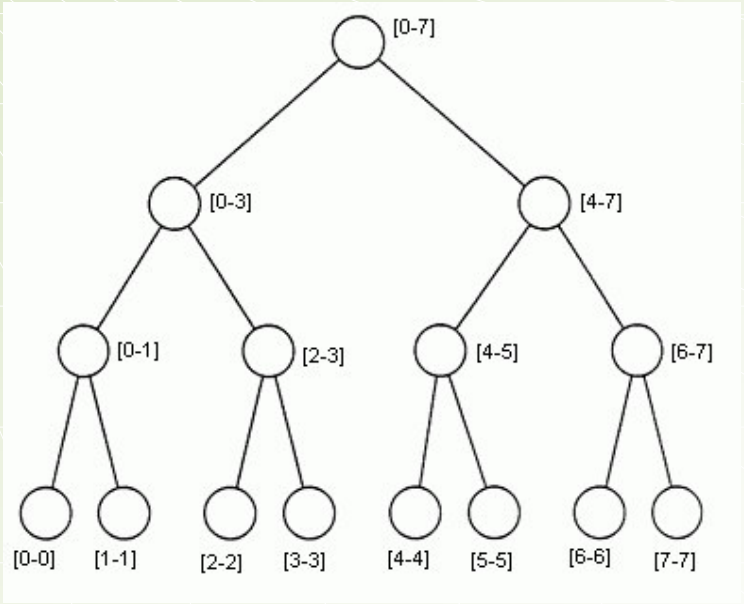
- Pogledajmo još jednom danu relaciju. Jedni dio koji je spor u njoj je računanje maksimuma elemenata iz nekog intervala. Kada bismo barem imali strukturu koja nam računa maksimum elemenata u intervalu... I to nije sve, ta struktura bi još morala podržavati i promjene elemenata u tom istom intervalu (nakon što izračunamo neki element, moramo ga zapisati u niz da bi ga koristili u daljnjim računima). Napravimo sažetak, treba nam struktura koja može:



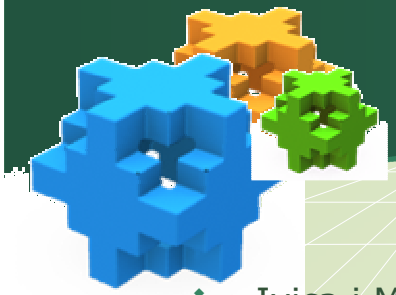
- Koja struktura to može? (Odgovor na sljedećem slideu)

Dinamika i strukture (nastavak)

- Odgovor je, naravno, interval/tournament stablo. Jedna od najfleksibilnijih, a opet jednostavnih struktura!



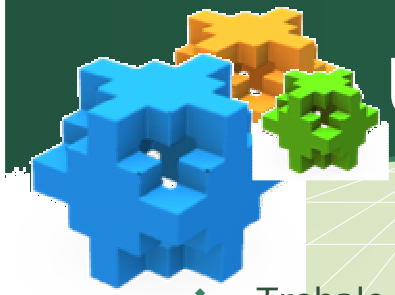
- Recimo da je relacija glasila - ***dp[x] = suma{ dp[x-D], dp[x-D+1], ..., dp[x-1] } + A[x]***. Zadatak se također može riješiti tournament/interval stablom, ali postoji i mnogo jednostavnije rješenje => Fenwickovo stablo/logaritamskom struktura (o kojoj ste također učili).



Uvod u teoriju igara

- Ivica i Marica igraju igru "Prirodnih brojeva i nule". Na početku je "*referentni broj*" upravo ***n (>1)***. Igra se igra tako da igrač koji je na potezu smanji referentni broj za 1. Pobjednik je onaj igrač koji prvi dođe do 0. Igru započinje Marica, te se nakon nje igrači naizmjenično mijenjaju.
  - Primjer igre za ***n=4***: Marica – "3"; Ivica– "2"; Marica – "1"; Ivica – "0" => Ivica je pobjednik!
  - Budući da igrači nemaju nikakav utjecaj na ovu igru, rješenje je trivijalno. Lako se pokaže da Marica pobjeđuje u slučaju neparnog ***n***, dok Ivica pobjeđuje kada je ***n*** paran.
    - Varijanta vrlo popularnog EciPeciPec...

- Učinimo sada prethodnu igru malo zanimljivijom: Recimo da onaj igrač koji je na potezu ima mogućnost biranja, može smanjiti referentni broj za 1 ili za 2. Sva ostala pravila ostaju ista.
  - Kažemo da je igrač (Ivica ili Marica) u poziciji ***X*** ukoliko je on/ona na potezu te je referentni broj upravo ***X***. Igra počinje tako da je Marica u poziciji ***n***. Jasno je da će onaj igrač koji je u poziciji 0 biti gubitnik (zato jer je suprotni igrač rekao navedeni broj).
  - Hoće li igrač u poziciji 1 biti gubitnik ili pobjednik?
    - Pobjednik, jer se može spustiti poziciju 0 i pobijediti. Skoro isto vrijedi i za poziciju 2.
  - Hoće li igrač u poziciji 3 biti gubitnik ili pobjednik?
    - Gubitnik, jer na koju se god vrijednost spusti (1 ili 2), protivnik ga može pobijediti spuštanjem na 0.
    - Primijetite da pretpostavljamo da i Ivica i Marica igraju "najbolje moguće" (tj. savršeno)
  - Lako se dođe do toga da je igrač u pozicijama 4 ili 5 pobjednik, u poziciji 6 gubitnik, u pozicijama 7 i 8 pobjednik, 9 -> gubitnik, itd.

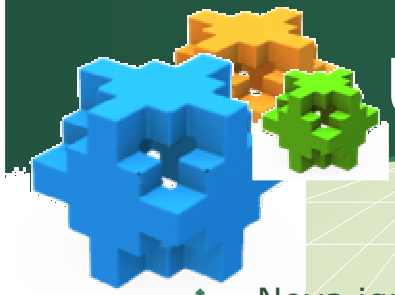


Uvod u teoriju igara (nastavak)

- Trebalo bi biti jasno da će pobjeda/gubitak nekog igrača ovisiti samo o tome na kojoj se poziciji on našao. Zbog toga je moguće svakom stanju (poziciji) pridijeliti jedno dodatno svojstvo, je li ta pozicija pobjednička ili gubitnička (hoće li igrač koji je u toj poziciji pobijediti ili izgubiti?)
  - Nacrtana je tablica koja govori za prvih nekoliko pozicija jesu li pobjedničke (P) ili gubitničke (G)...

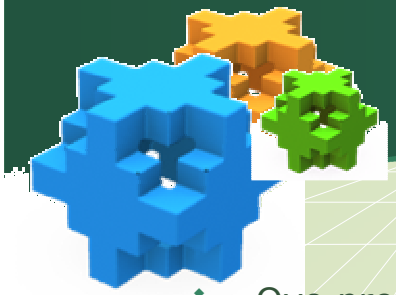
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**...**
G	P	P	G	P	P	G	P	P	...

- Dovoljno smo se upoznali sa zadatkom da bi dali neke definicije: poziciju ćemo nazvati pobjedničkom ukoliko vodi u barem jednu gubitničku poziciju, inače je zovemo gubitničkom.
  - Dakle pozicija je gubitnička ukoliko vodi isključivo u pobjedničke pozicije.
  - Pozicija 0 je pobjednička po naše definiciji igre. Iz toga je moguće dedukcijom odrediti za svaku drugu poziciju je li pobjednička ili ne.
  - Indukcijom se lako pokaže da je svaka pozicija djeljiva s 3 gubitnička, a sve ostale su pobjedničke
  - Kako ovo intuitivno objasniti? Ako je ***n=12***, Marica može reći brojeve 11 i 10. Ali neovisno o tome što je Marica rekla, Ivica može reći 9 (12-3). Nakon toga neovisno o Marici, Ivica može reći broj 9-3=6, i tako dalje sve dok ne dođe do 0 i pobijedi.
  - Opet vidimo da u neku ruku Marica uopće neće utjecati na igru (ukoliko je ***n*** djeljiv s 3)
    - Ako je n djeljiv s 3, kažemo da Ivica ima **pobjedničku strategiju.** U suprotnom pobjedničku strategiju ima Marica



Uvod u teoriju igara (nastavak)

- Nova igra: Ispred Ivice i Marica se nalaze dvije hrpe od po 100 kamenčića (svaka). Igrač koji je na potezu može iz **točno** jedne od te dvije hrpe izvući neki prirodni broj kamenčića (naravno, ne može izvući više kamenčića nego što ih ima u toj hrpi). Igrač koji više ne može vuči kamenčiće je gubitnik. Marica je prva na potezu. Tko ima pobjedničku strategiju?
  - Kako opisati poziciju na kojoj je igrač? Opisat ćemo je s parom brojeva ***(X,Y)*** koji označava da je u prvoj hrpi ostalo ***X***, a u drugoj ***Y*** kamenčića.
  - Koje su od tih pozicija pobjedničke, a koje gubitničke? Uočavamo da je pozicija ***(0,0)*** gubitnička (igrač na potezu više ne može izvući niti jedan kamenčić).
  - Rješenje: Svaka pozicija oblika (X,X) je gubitnička, dok su sve ostale pobjedničke. Iz stanja (5, 9) igrač može protivniku postaviti stanje (5,5), koje je gubitničko. U drugu ruku, stanje (5,5) vodi samo u pobjednička stanja u kojima su hrpe različite.
  - Situacija s 3 ili više hrpa tvori vrlo poznatu "Nim igru". Pobjednička strategija u takvoj igri je puno kompliciranija za otkriti i opisati.
  - Unatoč tome što je stanje neznatno kompliciranije: 3 broja ***(X,Y,Z)*** jedinstveno određuju poziciju.
- Dodatni zadaci iz teorije brojeva...



Dinamičko programiranje 3

- Ovo predavanje se još kvalificira kao uvod u dinamike. Materijala za daljnja predavanja na temu dinamika ne nedostaje:
- A-B prebrojavanja: Koja je suma znamenki svih častivih brojeva u intervalu [$4*10^{27}$, $169*10^{42}$] koji su djeljivi s 17. Broj je častiv ukoliko ne sadrži 3 uzastopne znamenke broja 6.
- Dinamike i linearna algebra: Na koliko načina možemo izgraditi neki (matematički) objekt. Dinamika nam pomaže da nađemo odgovor za relativno male ***n***. Linearna algebra pomaže da ubrzamo dinamiku za astronomske vrijednosti broja ***n***.
- Dinamike na stablima: praktički svaka dinamika na domeni jednodimenzionalnog niza se može poopćiti i rješavati na domeni stabla.
- Eksponencijalne dinamike: Većina algoritama koji rade u složenosti ***n!*n*** se mogu pomoću navedene metode spustiti na $n^2 2^N$.
- Duguljaste matrice: Kvazi-eksponencijalne dinamike koje rješavaju Minesweepera i većinu nerješivih *(oprezno s ovom klasifikacijom!)* problema na dvodimenzionalnim matricama za mnogo redova veličine brže od klasičnih rješenja.
- (Vrlo napredna tema!) Konveksnost i geometrijska dualnost: Što ako se trenutno stanje ne može izraziti kao funkcija prethodnih stanja, nego kao funkcija prethodnih i nekih trenutnih parametara? Za kakve je funkcije još moguće napraviti efikasnu strukturu?

Hvala na pažnji !

Ne zaboravite popuniti anketu (natprog.phpbb.net)

Pitanja?