1 (True or False) and Justify [20 bodov]

1.1 Zadanie

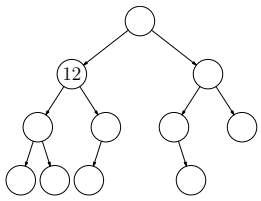
1. Pomocou spájaného zoznamu sa dá efektívne simulovať vektor.
2. Pomocou vektoru sa dá efektívne simulovať zásobník (stack).
3. Pomocou deque (obojsmernej fronty) sa dá efektívne simulovať vektor.
4. Pomocou setu (resp. multisetu) sa dá efektívne triediť – v čase $\Theta(n \log n)$.
5. Do hešovacej tabuľky veľkosti n ideme uložiť menej ako $n/2$ prvkov. Ak použijeme dobrú hešovaciu funkciu (napr. univerzálne hešovanie), s pravdepodobnosťou $> 1/2$ nenastanú žiadne kolízie.
6. Ak binárny vyhľadávací strom nie je vyvážený, tak nevieme zostrojiť jeho pre-order zápis v čase $O(n)$.
7. Daná je postupnosť n prvkov. Prvok, ktorý sa v nej najviackrát opakuje, vieme nájsť v čase $O(n \log n)$.
8. Zoznam n ľudí chceme usporiadať podľa dátumu narodenia. Najlepší algoritmus má časovú zložitosť $\Theta(n \log n)$.

1.2 Riešenie

1. FALSE. U vektoru vieme v konštantnom čase pristupovať k ľubovoľnému prvku (`operator[]`), zatiaľ čo u spájaného zoznamu to nejde.
2. TRUE. Pomocou metód `push_back` a `pop_back` vieme simulovať stack v amortizovanom konštantnom čase na operáciu.
3. TRUE. Existujú implementácie deque (napr. `deque` v C++), ktoré vedia všetko to, čo vektor, a ešte niečo naviac.
4. TRUE. Zoberieme prvky, vložíme do multisetu a potom ho prejdeme a vypíšeme ich v usporiadanom poradí.
5. FALSE. Pravdepodobnosť kolízie je omnoho väčšia, viď narodeninový paradox.
6. FALSE. Ľubovoľný proces, pri ktorom raz prejdeme celým stromom, vieme implementovať v $O(n)$ – totiž raz navštívime každý vrchol a nanajvýš dvakrát prejdeme každou z $n-1$ hrán. Vyváženosť na to nemá žiaden vplyv.
7. TRUE. Stačí ju napr. usporiadať a prejsť, alebo si v mape pamätať pre každý prvok počet výskytov.
8. FALSE. Vzhľadom na obmedzený (konštantný) rozsah možných dátumov narodenia bude efektívnejšie použiť CountSort, resp. niektorú z verzií BucketSortu.

2 Pohľad zvnútra: binárny vyhľadávací strom [3+7 bodov]

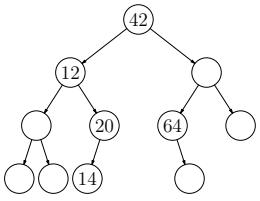
2.1 Zadanie



Na obrázku je binárny vyhľadávací strom.
a) Je vyvážený? Prečo?
b) Najmenšie prvky väčšie ako 12 sú 14, 20, 42 a 64. Doplňte ich do obrázka.

2.2 Riešenie

Strom je vyvážený: keď sa pozrieme na ľubovoľný vrchol, hĺbky jeho podstromov sa líšia najviac o 1.
Strom s doplnenými prvkami je na nasledujúcom obrázku:



(Najčastejšia chyba: dať 64 o vrchol vyššie.)

3 Pohľad zvnútra: optimalizácia výroby [15 bodov]

3.1 Zadanie

Kleofáš vyrába ručne maľované ťuplíky. Prieskum trhu ukázal, že ak ich bude predávať za cenu x, predá ich ročne $a/(x^b \ln(x+3))$. Ak Kleofáš ročne vyrobí t ťuplíkov, tak bude mať výrobné náklady $c + d/(t \ln(t+3))$ na jeden ťuplík. Pri riešení predpokladajte, že sa dá vyrobiť a predať aj neceločíselný počet ťuplíkov.
Napíšte program, ktorý načíta reálne čísla a, b, c, d (všetky sú väčšie ako 1) a hodnotu $\varepsilon > 0$ a s presnosťou ε vypočíta cenu $\leq 10^9$, za ktorú má Kleofáš predávať ťuplíky, ak chce mať maximálny možný čistý zisk.
Hint: Funkcia udávajúca celkový zisk v závislosti od ceny má na $[c, 10^9]$ jediné lokálne (a zároveň globálne) maximum.

3.2 Riešenie

Použijeme ternárne vyhľadávanie.

```
# globalne premenne: a, b, c, d, epsilon

def zisk(cena):
    # spocita zisk pre danu cenu za kus
    kusov = a / pow(cena,b) / log(cena+3)
    naklady1 = c + d / kusov / log(kusov+3)
    zisk1 = cena - naklady1
    return kusov * zisk1

lo, hi = c, 10**9
while hi-lo > epsilon:
    m1, m2 = (2*lo+hi)/3., (lo+2*hi)/3.
    if zisk(m1) > zisk(m2):
        hi = m2
    else:
        lo = m1

print "Optimalna cena je priblizne: ", lo
```

4 Pohľad zvnútra: náhodný výber [5+5+(5+bonus) bodov]

4.1 Zadanie

a) V pamäti je pole $A[0..n-1]$. Napíšte program, ktorý ho náhodne preusporiada.
b) V pamäti je pole $A[0..n-1]$ a číslo k ($k < n$). Napíšte program, ktorý náhodne vyberie k prvkov poľa A. (Pole A je dovolené meniť.)
c) Na vstupe bude postupne prichádzať n prvkov. Napíšte program, ktorý použije len $O(1)$ pamäte a náhodne vyberie jeden z nich. Bonusové body za všeobecnejšie riešenie: pomocou $O(k)$ pamäte vybrať k-ticu náhodných prvkov.
(Preusporiadania a výbery musia byť **rovnomerne** náhodné – všetky možnosti musia byť rovnako pravdepodobné.)

4.2 Riešenie podúlohy a)

Postupne ideme poľom a pre každé políčko náhodne vyberieme, ktorý prvok naň umiestnime. Časová zložitosť $\Theta(n)$.

```
for i in range(len(A)):
    x = randint(i,len(A)-1) # nahodne cislo z [i,len(A)-1]
    if x>i:
        A[i], A[x] = A[x], A[i]
```

Iné riešenie (akoby inverzné k predchádzajúcemu): Postupne pridávame prvky. Vždy, keď pribudne nový, vyberieme mu náhodnú pozíciu a umiestnime ho na ňu.

```
for i in range(len(A)):
    x = randint(0,i)
    if x<i:
        A[i], A[x] = A[x], A[i]
```

4.3 Riešenie podúlohy b)

Stačí zobrať prvé riešenie predchádzajúcej podúlohy a spraviť v ňom len prvých k iterácií. V tomto okamihu je na prvých k pozíciách poľa A náhodná k-tica jeho prvkov.

```
for i in range(k):
    x = randint(i,len(A)-1) # nahodne cislo z [i,len(A)-1]
    if x>i:
        A[i], A[x] = A[x], A[i]
return A[:k]
```

4.4 Riešenie podúlohy c)

Aj majster tesár sa utne :) tak aj mne sa podarilo napísať zadanie, ktoré sa dalo interpretovať tak, že n je vopred známe. V takomto prípade je riešenie ľahké: vygenerujeme jedno náhodné číslo x od 1 po n, načítame x prvkov a vypíšeme posledný načítaný.

V prípade, že n nie je vopred známe, ide o úlohu z prednášky. Riešenie pre $k = 1$: počas čítania prvkov si pamätáme jeden prvok – kandidáta. Na začiatku načítame prvý prvok, ten bude kandidátom. Potom postupne čítame ďalšie prvky. Keď prečítame i-ty prvok, s pravdepodobnosťou $1/i$ ho zoberieme za nového kandidáta, inak ho zahodíme.
Indukciou sa dá dokázať, že po spracovaní i-teho prvku má každý z prvých i prvkov pravdepodobnosť $1/i$, že je práve kandidátom.

5 Pohľad zvonka: rozvrh [20 bodov]

5.1 Zadanie

Na matfyze sa v budúcom semestri bude prednášať n prednášok. Každá má presne stanovený čas začiatku $Z[i]$ a konca $K[i]$ (pre jednoduchosť nech sú to prirodzené čísla udávajúce offset od začiatku týždňa). Taktiež pre jednoduchosť predpokladajme, že každá prednáška môže byť robená v každej miestnosti.
Ako pseudokód alebo kus programu napíšte čo najefektívnejší algoritmus, ktorý zistí, koľko najmenej rôznych miestností treba na to, aby sa všetky prednášky mohli uskutočniť. (Stačí zistiť počet miestností, netreba vypisovať konkrétne rozmiestnenie predmetov.) Nezabudnite na dôkaz správnosti a odhad časovej zložitosti.

Príklad: Ak majú byť prednášky v časoch $[30, 70]$, $[20, 80]$, $[72, 95]$ a $[65, 73]$, tak treba 3 miestnosti (prvá a tretia prednáška môžu byť po sebe v tej istej miestnosti).

5.2 Riešenie

Nech k je najväčšie číslo také, že v každom okamihu beží nanajvýš k prednášok naraz.
Z definície k je zjavné, že potrebujeme aspoň k miestností.
Na druhej strane, ľahko dokážeme, že k miestností stačí. Predstavme si, že začne bežať týždeň. Vždy, keď má začať niektorá prednáška, nájde jej prednášajúci prvú voľnú miestnosť a tú použije. Keďže nikdy nebeží viac ako k prednášok naraz, vždy, keď začína prednáška, je aspoň jedna z miestností voľná, takže každému sa podarí nájsť pre seba miestnosť.
Potrebujeme teda spočítať číslo k. To vieme spraviť napr. nasledovne: usporiadame si všetky začiatky aj konce prednášok. V usporiadanom poradí potom cez ne prejdeme, pričom si udržiavame počítadlo, koľko prednášok práve beží. Toto riešenie má kvôli triedeniu časovú zložitosť $\Theta(n \log n)$.
Iné riešenie: S rovnakou časovou zložitosťou vieme aj zostrojiť jeden možný rozvrh. Stačí mať napr. miestnosti usporiadané v prioritnej fronte podľa času, kedy sa uvoľnia. Potom postupne spracúvame prednášky usporiadané podľa času začiatku. Ak je niektorá zo skôr použitých miestností už voľná, použijeme ľubovoľnú takú miestnosť. A ak sú všetky plné, pridáme novú a dáme aktuálne spracúvanú prednášku do nej.

6 Pohľad zvonka: hokej [4 × 5 bodov]

6.1 Zadanie

Hokejový zápas sa skončil 4:1. Ľuboško by chcel zrekonštruovať priebeh zápasu. Pamätá si len to, že nikdy nebol stav 3:0 ani 0:1. Koľko rôznych priebehov zápasu vyhovuje tomu, čo si Ľuboško pamätá?
(Odpoveď je 2: buď dali hostia gól za stavu 1:0, alebo za stavu 2:0.)
Budeme riešiť všeobecnú verziu tejto úlohy. Cieľom je zistiť počet možných priebehov zápasu pre dané záverečné skóre $d : h$, číslo n a polia $A[0..n-1]$ a $B[0..n-1]$. Pre každé i si je Ľuboško istý, že stav nikdy nebol $A[i] : B[i]$.

a) Napíšte (ako pseudokód alebo kus programu) rekurzívny algoritmus, ktorý vyskúša a overí všetky možné priebehy zápasu a spočíta, koľko z nich vyhovuje. (Odporúčanie: začne tým, že vyskúša obe možnosti, kto strelil posledný gól zápasu.)
b) Pridajte do predchádzajúceho algoritmu memoizáciu tak, aby vznikol algoritmus s časovou zložitosťou polynomiálnou od d, h, n. Odhadnite jeho časovú zložitosť.
c) Uveďte ekvivalentný algoritmus, ktorý túto úlohu rieši pomocou dynamického programovania.
d) Zistite, koľko rôznych priebehov mohol mať zápas, ktorý skončil 4:6, pričom nikdy nebolo skóre 2:2 ani 1:5. (Ľubovoľný postup vedúci k správnej odpovedi je OK. Jednou z možností je simulovať jeden z algoritmov b), c).)

6.2 Riešenie podúlohy a)

Podľa návodu v zadaní: skúšame od konca všetky možné priebehy zápasu a zarátame každý, kedy sa nám podarí (bez toho, aby sme prešli cez zakázané skóre) dostať sa až späť k stavu 0:0.

```
def priebehy(d0,h0):
    for d1,h1 in zip(A,B):
        if (d0,h0)==(d1,h1): return 0
    if d0==0 and h0==0: return 1
    odpoved = 0
    if d0>0: odpoved += priebehy(d0-1,h0)
    if h0>0: odpoved += priebehy(d0,h0-1)
    return odpoved

print priebehy(d,h)
```

O chlp lepšie riešenie je neskúšať zbytočne zakaždým všetky zakázané skóre. Keď si ich nahádžeme do vhodnej dátovej štruktúry (napr. hashsetu), prípadne keď nimi vhodne vyplníme dvojrozmerné pole boolovských hodnôt, dostaneme efektívnejšie riešenie.

```
def priebehy(d0,h0):
    global zakazane
    if (d0,h0) in zakazane: return 0
    if d0==0 and h0==0: return 1
    odpoved = 0
    if d0>0: odpoved += priebehy(d0-1,h0)
    if h0>0: odpoved += priebehy(d0,h0-1)
    return odpoved

zakazane = set()
for d1,h1 in zip(A,B): zakazane.add( (d1,h1) )
print priebehy(d,h)
```

6.3 Riešenie podúlohy b)

Pridáme memoizáciu. Ak to spravíme s prvým vyššie uvedeným riešením, dostaneme časovú zložitosť $O(dhn)$, lebo pre každé z $(d+1)(h+1)$ možných skóre kontrolujeme všetkých n zakázaných stavov. Z druhého rekurzívneho riešenia vznikne riešenie s očakávanou časovou zložitosťou $O(dh)$.

```
def priebehy(d0,h0):
    global zakazane, memo
    if (d0,h0) in zakazane: return 0
    if d0==0 and h0==0: return 1
    if (d0,h0) in memo: return memo[ (d0,h0) ]
    odpoved = 0
    if d0>0: odpoved += priebehy(d0-1,h0)
    if h0>0: odpoved += priebehy(d0,h0-1)
    memo[ (d0,h0) ] = odpoved
    return odpoved

memo = {}
zakazane = set()
for d1,h1 in zip(A,B): zakazane.add( (d1,h1) )
print priebehy(d,h)
```

Ak by sme použili namiesto hashsetu na zakázané stavy a hashmapy na memoizáciu dvojrozmerné polia, mali by sme dokonca zaručenú časovú zložitosť $O(dh)$.

```
def priebehy(d0,h0):
    global zakazane, memo
    if zakazane[d0][h0]: return 0
    if d0==0 and h0==0: return 1
    if memo[d0][h0] != -1: return memo[d0][h0]
    odpoved = 0
    if d0>0: odpoved += priebehy(d0-1,h0)
    if h0>0: odpoved += priebehy(d0,h0-1)
    memo[d0][h0] = odpoved
    return odpoved

memo = [ [ -1 for h0 in range(h+1) ] for d0 in range(d+1) ]   # pole (d+1)x(h+1) obsahujuce same -1
zakazane = [ [ False for h0 in range(h+1) ] for d0 in range(d+1) ]
for d1,h1 in zip(A,B): zakazane[d1][h1] = True
print priebehy(d,h)
```

6.4 Riešenie podúlohy c)

Ukážeme si implementáciu, ktorá je čo najviac identická s tou pri memoizácii rekurzie:

```
zakazane = [ [ False for h0 in range(h+1) ] for d0 in range(d+1) ]
for d1,h1 in zip(A,B): zakazane[d1][h1] = True

memo = [ [ -1 for h0 in range(h+1) ] for d0 in range(d+1) ]
```

```
for d0 in range(d+1):
    for h0 in range(h+1):
        if zakazane[d0][h0]:
            memo[d0][h0] = 0
            continue
        if d0==0 and h0==0:
            memo[d0][h0] = 1
            continue
        odpoved = 0
        if d0>0: odpoved += memo[d0-1][h0]
        if h0>0: odpoved += memo[d0][h0-1]
        memo[d0][h0] = odpoved

print memo[d][h]
```

6.5 Riešenie podúlohy d)

Simulujeme riešenie podúlohy c), pričom postupne vyplníme dvojrozmernú tabuľku. Mali by sme sa dopracovať k výslednému počtu 96 možných priebehov.
Iné riešenie: Všetkých priebehov zápasu je $\binom{4+6}{4} = 210$. Tých, ktoré obsahujú stav 2:2, je $\binom{2+2}{2}\binom{2+4}{2} = 6.15 = 90$. Tých, ktoré obsahujú stav 1:5, je $\binom{1+5}{1}\binom{3+1}{3} = 6.4 = 24$. Žiaden zápas nemôže obsahovať aj stav 2:2, aj stav 1:5. Preto je dobrých priebehov zápasu $210 - 90 - 24 = 96$.