NIO 2020 finale

Løsningsforslag

Norveksit

Et punkt er i Norvestige hvis Y > X

Et punkt er i Sørøstia hvis X > Y

Et punkt ligger på grensen (og tilhører ingen av landene) hvis X = Y.

Kan teste punktene i hver by for å se hvilket land de er i

Norveksit

Kan teste alle punkter i byene

Kan teste kun punktene som ligger på kanten av byene

Holder å teste nordvestligste og sørøstligste hjørne

```
bool iNordVest = N > V;
bool iSørØst = Ø > S;
if (iNordVest && iSørØst) feil = true;
else if (iNordVest) antallNV++;
```

Rettferdig Fordeling

Likner på en 0-1 knapsack oppgave

Problemet er at det er 2 personer som kan få kort

Kan holde styr på alle mulige summer av verdi på kortene til Lise og alle mulige summer av verdi på kortene til Lotte som det er mulig å oppnå med de første k kortene

Dette vil kreve $O(N^3 \times 2500^2)$ tid - ikke gjennomførbart, selv sannsynligvis ikke en gang for første subtask.

Rettferdig Fordeling

Første subtask: $N \leq 15$

Kan prøve alle mulige kombinasjoner - gi kort til Lise, Lotte eller ingen.

Kjøretid $O(3^N)$

Rettferdig Fordeling

Andre subtask: $N \leq 26$

Kan gjøres med meet-in-the-middle: Prøv alle mulige fordelinger med de første N/2 og alle mulige fordelinger med de siste N/2.

Hvis Lise har en overvekt på X i en fordeling av de første kortene, så må den kombineres med en fordeling hvor Lotte har en overvekt på X blant de siste kortene.

Kjøretid $O(3^{N/2})$

Rettferdig Fordeling

Tredje subtask: Bare toerpotenser

Kan løses grådig

Ta de største kortene først og fordel så jevnt som mulig.

Hvis oddetall, se om det er mulig å kompensere med de resterende kortene.

Kjøretid O(N log N)

Rettferdig Fordeling

Det som er viktig er ubalansen mellom Lise og Lotte.

best[Nx2500 +1][N+1]
La best[A][K] være den største mulige totalverdi på kort som er mulig å gi bort blant de K første kortene, slik at den som har mest har nøyaktig A kroner mer enn den andre.

Når man betrakter et nytt kort, så kan man enten gi til den som allerede har mest, den som har minst, eller la være å gi den til noen.

Svaret blir da best[0][N] / 2. Kjøretid $O(N^2 \times 2500)$

Snømenn

Grådig løsning??

Hvordan vet vi om en snøball skal begynne en ny snømann eller om den skal legges oppå en tidligere snømann eller om den ikke skal inngå i løsningen?

Lemma: Dersom det finnes en løsning med K snømenn, så finnes det en løsning hvor de K største er “føtter” og de K minste er “hoder”

Snømenn - bevis for lemma

La oss anta at det finnes en løsning med K snømenn hvor de K minste ikke er hoder. Da finnes det en snøball **d** som ikke er blant de K minste som fungerer som et hode.

Hvis det finnes en snøball blant de K minste som ikke er i bruk, så kan vi bytte ut **d** med denne, og få en løsning som bruker flere av de K minste som hoder.

Hvis det ikke gjør det, så finnes det minst én snøball blant de K minste som har rollen som “mage”

Snømenn - bevis for lemma

Kaller operatoren $\lll$ for “mye mindre enn”
Def: $X \lll Y$ hvis $X \leq Y \div 2$

La **b** være en mage blandt de K minste. Vi har da to snømenn

a $\lll$ **b** $\lll$ **c**
d $\lll$ **e** $\lll$ **f**
hvor **b** < **d**

Snømenn - bevis for lemma

$a \lll b \lll c$
$d \lll e \lll f$
$b < d$

Disse snømennene kan bygges om til

$a \lll d \lll \max(c, e)$
$b \lll \min(c, e) \lll f$

$a \lll b < d \Rightarrow a \lll d$
$d \lll e \leq \max(c, e) \Rightarrow d \lll \max(c, e)$
$b \lll c$ og $b < d \lll e \Rightarrow b \lll c$ og $b \lll e \Rightarrow b \lll \min(c, e)$
$\min(c, e) \leq e \lll f \Rightarrow \min(c, e) \lll f$

Snømenn - bevis for lemma

Det finnes en løsning med K snømenn hvor de K minste ikke er hoder ⇒
Det finnes en løsning med K snømenn som bruker flere av de K minste som hoder (og som ikke har mindre føtter enn den opprinnelige løsningen) ⇒
Det finnes en løsning med K snømenn hvor de K minste er hoder

Via symmetri har vi at det finnes en løsning hvor de K minste er hoder og de K største er føtter

Snømenn

Binærsøk etter K. For en gitt K vet vi hvilke snøballer som skal være hoder og hvilke som skal være føtter.

Kan vise at det er mulig å alltid ha det minste hodet over de minste føttene, osv.

Ta de minste hodene først. Grådig plasser en så liten snøball som mulig som mage mellom hodet og den tilhørende foten. Feiler hvis vi ikke har noen snøball som er stor nok for hodet eller om den minste snøballen vi kan bruke er for stor for de tilsvarende føttene.

Søketre

Må implementere funksjoner for å bygge et søketre ut fra nodesekvensen.

Søketre

```
class Tree {
	public Tree(int value) {
		this.value = value;
	}
	int value;
	Tree left;
	Tree right;
}

//sett inn value i treet, og returner hva som er den nye rotnoden
Tree insert(Tree root, int value) {
	if (root == null) return new Tree(value);
	if (value < root.value) root.left = insert(root.left, value);
	else root.right = insert(root.right, value);
	//rot-noden er uendret
	return root;
}
```

Søketre

For første subtask er det kun maks 7!=5040 mulige sekvenser, så vi kan raskt teste ut alle.

For andre subtask er det for mange sekvenser til å prøve alle, men ved å gjøre litt pruning kan man få det til å gå innenfor kjøretiden.

Hold oversikt over hvilke tall som det til enhver tid er lov å sette inn i treet. Opprinnelig kun rotnoden i resultat-treet, men barn til noder du setter inn blir lov.

Søketre

La T være et ikke-tomt tre med V og H som barn.

For å lage T må man først sette inn tallet i roten til T

Alle tall som skal inn i V må komme i en rekkefølge som lager treet V

Alle tall som skal inn i H må komme i en rekkefølge som lager treet H

Søketre

Tall som skal til V og tall som skal til H kan komme fritt om hverandre

Blå sirkel er tallet som skal være i noden til T
Grønne sirkler skal til V
Lilla sirkler skal til H

Søketre

Det er nCr(|V|+|H|,|V|) måter å velge hvilke av de |V|+|H| tallene som skal være tallene som skal til V
nCr(A,B) = A!/(B!×(A-B)!)

Hvis χ(T) er antall måter å lage treet T på så har vi altså

$$\chi(T) = \chi(V) \times \chi(H) \times nCr(|V|+|H|,|V|)$$

χ(V) og χ(H) kan regnes ut ved rekursjon

nCr(|V|+|H|, |V|) (modulo 200 009) kan regnes ut med Pascals trekant. Andre metoder vil gi problemer med overflow.

Stormannsgalskap

Subtask 1
K = 10×U, D =0

Siden D er 0 går alltid stormannsgalskap ned. Det er dermed enten mulig å bli ferdig på dag 1 eller aldri mulig å bli ferdig.

Hvis noen av de direkte underordnede til Fredrik har flere enn 10 underordnede så er det umulig.

Hvis ikke er svaret 1.

Stormannsgalskap

Subtask 2
N ≤ 100, Stormannsgalskap faller til 0 etter hver dag

Gjør kunngjøringer helt til neste kunngjøring vil gi for mye stormannsgalskap. Sett deretter all stormannsgalskap til 0 og begynn ny dag.

Hvis det ikke er mulig å gjøre noen kunngjøring en dag så er det UMULIG. Hvis ikke er svaret max 100, så vi kan simulere en etter en dag.

Stormannsgalskap

Subtask 3
$N \leq 500$, $U = D$

Begrensningene gjør at det kan ta maks 500 dager, så en ganske direkte simulering fungerer fint.

Stormannsgalskap

Subtask 4
N ≤ 100 000, Treet er ikke dypt

Kan ikke simulere alle dager. Beregn i stedet hvor mange dager man må vente før neste forhold kan legges til.

Krever også mye tid å oppdatere galskaper. Kan løses ved å holde styr på hvilke personer som har galskap > 0 og kun oppdatere disse når man venter.

Stormannsgalskap

Subtask 5

Grafen er til en hver tid en skog - en samling med trær.

Trær forbindes for å lage nye trær, helt til skogen bare inneholder ett stort tre.

Roten i et tre er alltid den mest gale. Altså trenger vi bare å bekymre oss om folk som er røtter.

Union-find holder styr på hvem som er roten i et gitt tre.

Stormannsgalskap

Når vi slår sammen to trær trenger vi bare å passe på at roten i det nye treet ikke blir for gal.

For hver rot hold styr på hvor mange noder som treet inneholder. Når en ny kant gjør at vi legger til et tre under et annet så forteller størrelsen på under-treet hvor mye mer galskap som kommer til å bli lagt til på roten av over-treet.

Stormannsgalskap

Vi trenger ikke å oppdatere galskapen til alle røtter hver gang vi venter en eller flere dager.

Hvis vi lagrer unna både tidspunkt for når sist galskapen økte og hvor mye den da ble, så kan vi i stedet retroaktivt beregne hvor mye de har roet seg ned.