Lösningsförslag Final 2018

Rulltrappa

För fullt poäng behöver man korrekt beräkna hur lång tid det tar att åka längs med vardera rulltrappa. Detta består av två delar: tiden för kön att försvinna och tiden att åka upp. Detta blir $\frac{l}{a} + \frac{M}{g}$ för vänstra sidan och $\frac{r}{b} + \frac{M}{s}$ för högra sidan. Därefter beövs dessa tider endast jämföras för att avgöra vilken som är snabbast (de behöver inte vara heltal).

Flourtanten

Delpoäng 1: för denna gruppen räcker en $O(n^2)$ lösning. Detta kan göras väldigt enkelt genom att testa att flytta Björn till varje av de n möjliga platserna i kön, räkna ut glädjen för placeringen och skriva ut den största glädjen av alla dessa.

Delpoäng 2 och 3: Den viktiga insikten här är att om sekvensen är icke-ökande/sjunkande så kommer resultatet alltid att förbättras/försämras beroende på riktningen Björn flyttar sig, vilket innebär att glädjen maximeras om han står i slutet eller början av kön. Då räcker det att skriva testa att placera i slutet/början och skriva ut vilken som ger mest glädje. Svaret för dessa grupperna får inte alltid plats i 32-bitars integers.

Full poäng: För att få full poäng kan vi kolla lite noggrannare på vad som händer när Björn byter plats med någon bredvid honom. Om Björn befinner sig på plats i och byter med person $i+1$ kommer den att bidra med $happiness[i+1]*(i)$ istället för $happiness[i+1]*(i+1)$. Om glädjen för den ursprungliga kön, g, förberäknas blir alltså nya glädjen $g + happiness[i+1]*(i) - happiness[i+1]*(i+1)$. Om Björn byter plats med person $i-1$ gäller samma argument. Detta tillåter oss att besöka varenda möjlig konfiguration i $O(n)$ tid genom att göra 2 loops, en där han flyttas till alla platser höger och en till alla platser till vänster. En potentiell bug är att man glömmer att den urpsrungliga konfigurationen kan vara optimal.

Trevlig väg

Delpoäng 1: Eftersom det finns få kanter sammanlagt kan problemet lösas genom att pröva varenda möjliga väg. Detta kan göras enkelt genom en rekursiv funktion som branchar vid varje kant.

Delpoäng 2: För att lösa denna grupp går det att använda dynamisk programmering i $O(n^2)$. Det går inte att avgöra lokalt vid varje nod hur många noder det är optimalt att ha besökt innan. Därmed blir staten index på nod och hur många som besökts innan, och det är optimalt att girigt maximera längden på vägen till varje state. Transitionen blir då $dp[u][dist] = max(dp[k][dist-1] + w)$, där k är en förälder till u och w är vikten på kanten från k till u. Ordningen som dessa statesen evalueras hade kunnat varit knepigt, då vi vill garantera att alla noder som är förfäder till en given nod besöks innan den. Det visar sig dock att en topologisk sortering garanterar exakt detta! Därmed, om vi transitionar framåt längs med alla kanter från $i = 0$, sedan $i = 1, \ldots, i = n-1$. kommer det att bli rätt ordning. Att notera är att längden på vägarna lätt kan overflowa en 32-bitars integer.

Full poäng

För full poäng kan ovanstående DP optimiseras. Om vi betraktar uttrycket för glädjen av en exempelväg kan denna se ut som följande $\frac{4+5+2}{3}$. Låt oss kalla detta medelvärdet för k. Vi kan då knacka lite algebra:

$$\frac{4+5+2}{3} = k \tag{1}$$

$$4+5+2 = 3k \tag{2}$$

$$(4-k)+(5-k)+(2-k) = 0 \tag{3}$$

Vad som är fint med detta är att om vi vet vad k är, så kan vi helt enkelt ändra vikten på varje kant från $w[e]$ till $w[e]-k$ och hitta längsta vägen i linjär tid. Om denna är positiv betyder det att man kan få medelvärdet k. Detta kanske känns ohjälpsamt, då k är ett flyttal. Man kan dock insé att vi kan binärsöka över k. Detta beror på att om det går att ett medelvärde k så kan man uppnå alla medelvärden mindre än k, men inte större än k. Transitionen blir då $dp[u] = max(dp[v] + w - k, dp[u])$, där v är en förälder till u och w är vikten på kanten från v till u.

Teleportgång

Delpoäng 1: För denna gruppen räcker det att simulera den optimala strategin ett flertal gånger. Denna består av att teleportera dig om du är av avstånd D till målet, annars gå till det. Noder som inte kan ta sig till målet alls har avstånd infinity (stort tal). För att räkna ut avstånden från varje nod till målet kan vi göra en BFS från det. Eftersom vi inte vet vad D är kan vi bruteforcea

vad den kan vara (upp till n). För varje D gör vi sedan en godtycklig mängd simulationer (10^5 fick accepted med c++). För att underlätta med kodningen kan det hjälpa att veta att väntevärde innebär ungefär lika med medelvärdet över en oändligt stor urvalsgrupp. För att uppskatta det kan man därmed räkna det som vanligt medelvärde, d.v.s. antalet steg totalt delat med antalet simulationer (för varje D).

Delpoäng 2: För denna grupp hinner vi inte simulera. Istället kan vi beräkna väntevärdet för varje D i linjär tid. Här hjälper det mycket att tänka på vårt svar som ett medelvärde istället för det givna uttrycket för väntevärde. Om man försöker räkna ut exakta sannolikheterna p_1, p_2 etc. blir det snabbt krångligt. Istället kan svaret delas upp i 2 delar: genomsnittlig mängd teleporteringar och genomsnittlig mängd kanter korsade. Låt c vara antalet noder av avstånd D till målet. Då kommer vi teleportera tills vi får ett gynsammt utfall av sannolikheten $\frac{c}{n}$. Det visar sig att genomsnittliga antalet prövningar för ett gynsammt utfall av sannolikhet p är $\frac{1}{p}$, därmed teleporterar vi gensomsnittligt $\frac{n}{c}$ gånger. För att räkna ut antalet kanter vi korsar i gensomsnitt använder vi faktumet att man hamnar på varje nod lika ofta. Låt summan av alla avståndet till alla noder av avstånd mindre än D till målet vara t. Genomsnittliga antalet kanter som korsas blir då $\frac{t}{c}$. Detta kan naivt implementeras i $O(n^2)$ tid. En potentiell bug är att man glömmer att kolla om det är snabbare att gå in i målet direkt.

Full poäng

För full poäng behöver det inte göras jättemycket mer. Ett sätt är att räkna ut väntevärdet för varje D i konstant tid genom att återanvända beräkningar från förra D. Detta beror på att vårt svar egentligen bara beror på två variabler: c och t. När vi ökar D ökar dessa, därmed kan vi spara c och t för gamla D. Det går också att tenärsöka över D, dock är detta troligtvis jobbigare.

1 Vilse i tidtabellen

Full poäng

Detta är ett relativt jobbigt implementeringsproblem med flera hörnfall. Den första observationen man kan göra är att svaret max kan innehålla m tider. Detta är för att för varje tid på busstabellen så finns det endast en tid som är t_0 sekunder innan den. Om man räknar ut tiden det skulle ta att checka alla dem här m busstiderna med de n tiderna tills en buss anländer, så skulle lösningen redan få TLE. Detta måste betyda att det finns ett knep som kan göra detta snabbare. Observationen man kan göra är att det går att reducera n från 10^6 till 86400. Detta beror på att busstiderna efter de första 24 timmarna måste vara periodiska med de första. Detta innebär dock inte att man bara behöver betrakta busstider med $t < 86400$, för att de senare tiderna kan vara felaktiga. Därmed får man även loopa igenom de och kolla att de följer en exakt period

av de första 24 timmarna och att det inte "saknas" några busstider mellan flera dygn (exempelvis är de nästkommande tiderna $1, 86400 * 2 + 1$ ogiltiga trots de är periodiska). Testdatan är dock svag, så det är inte obligatoriskt att checka det här. Det är också uppenbarligen fel om det inte finns några busstider de första 24 timmarna. Därefter måste vi kolla vilka tider som stämmer med schemat. Sträng-formatet på strängarna måste först omvandlas till heltal så att man lätt kan arbeta med dem. Omvandlingen är ganska lätt då man bara kan multiplicera antalet sekunder, minuter eller timmar med antalet sekunder som går i en sekund, minut eller timme.

Hur man ska lösa vilka tider som är möjliga kan göras på flera olika vis. Ett sätt som undviker flera jobbiga hörnfall är att ändra alla $t_i < 86400$ till $t_i = t_i - t_{i-1}, (i > 0)$. För att kunna använda denna avstånds lista till nästa buss så måste vi även skapa en lista som givet en tid kan berätta hur långt det är till nästa buss. Skapandet och uträkningen av detta kan göras linjärt. Sedan kan man loopa igenom alla 86400 start positioner och titta för alla $t_i < 86400$ ifall avståndet till den nästkommande bussen är just M_i. Efter att man har kollat t_i så flyttar man fram sin tid med t_i så att kan kan titta på nästa tid. För att inte få TLE så måste man om avståndet inte stämmer gå ut ur loopen så att tiden inte blir 86400^2. I den här uträkningen så stöter man också på några hörnfall. Exempelvis så vill man ifall det är den första bussen upptäcka ifall man står på en busstid, men inte annars då den i så fall alltid skulle säga att avståndet till nästa buss är 0 sekunder. Detta får man lösa genom att låtsas att man är en sekund längre fram och sedan lägga till en sekund på avståndet till nästa buss. Här måste man uppenbarligen även använda den nuvarande tiden mod 86400 då tiden kan gå över till nästa dygn. Till sist så skriver man fel ifall ingen tid hittades som stämmde med t eller om vissa tider i t är felaktiga. Annars så skriver man vilka tider som stämmde ihop med t. Den slutgilitiga koplexiteten på denna lösning blir O(sekunder på ett dygn * m).

Cirkelskivsvärlden

Under den originella tävlingen var testdatan svag och det räckte att placera all magi precis bredvid målet. Vi eftersöker därmed någon som hade kunnat designa roliga testfall.