Mehir Seyum Wolde
CINTE3

Uppgift 2 Antal sätt att producera ett prov

2.1 Konstruera en algoritm som givet *N*, *x* och *u*[1..*N*] beräknar antalet sätt som jobbet kan fördelas på.

Algoritmbeskrivning:

Iden till denna lösning kommer från liknande dynamiska programmeringsproblem. Algoritmen för uppgift två löser problemet med dynamisk programmering för att slippa genomföra liknande beräkningar flera gånger om. Med hjälp av en matris $A_{n,x+1}$ där varje rad *i* representerar varje tal *u[i],* det vill säga hur många uppgifter lärare *i* kan göra. Varje kolumn representerar det antal uppgifter som kan utföras enda upp till *x*, det vill säga intervallet *[0,...,x]*. Detta betyder att på rad *i* kolumn *x* finner man antal sätt som jobbet kan fördelas på bland dem *i* första lärarna i *u* med *x-1 uppgifter*.

Exempel med u = {1,3,4}, x = 2, n = 3

Man börjar med att initiera en tom matris $A_{n,x+1}$ (matris i detta exempel noll indexeras). Efter detta steg fyller man första raden (*i* = 0) i matrisen med 1 eller 0, eftersom det endast finns 1 eller 0 sätt att bilda talet *j* med delmängden *u[0]* vilket beror på om *u[i]* är större än/lika kolumnvärdet *j* eller mindre än det. Efter detta fyller man även första kolumnen, $A_{(0...n-1),0}$ med 1 eftersom det endast finns ett sätt att få summan 0, vilket är genom att exkludera alla element från mängden.

i/j	0	1	2
0			
1			
2			

→

i/j	0	1	2
0	1	1	0
1			
2			

→

i/j	0	1	2
0	1	1	0
1	1		
2	1		

Efter detta börjar man på $A_{1,1}$, vilket i detta fall kontrollerar alla möjliga sätt för mängden *{u[0],u[1]}* att bilda summan 1. I detta fall kan vårt tal *u[1]* anta alla värden *0,..,u[i]*,vilket nu är 0,1,2,3 och eftersom att detta är större än kolumnvärdet 1 kommer vi begränsas till endast 1 och 0. För att få detta antal måste man då kontrollera på hur många sätt man kan bilda restsumman vid dem två fallen med mängden {*u[0]*} vilket vi gör genom att använda de existerande värdena i matrisen. Algoritmen upprepar även detta för resten av rad 1 vilket endast består av $A_{1,2}$. Här är *u[i]* = 3 fortfarande större än kolumnvärdet så det begränsas till att anta värdena 0,1,2 vilket man får genom att addera alla värdena på raden över.

i/j	0	1	2
0	1	1	0
1	1	1+1	
2	1		

→

i/j	0	1	2
0	1	1	0
1	1	2	1+1
2	1		

→

i/j	0	1	2
0	1	1	0
1	1	2	2
2	1		

…..

Mehir Seyum Wolde
CINTE3

i/j	0	1	2
0	1	1	0
1	1	2	2
2	1	3	5

Algoritmen upprepar denna logik enda tills man når $A_{n-1, x}$ vilket kommer innehålla det sökta värdet man letar efter. Alltså på plats $A_{n-1, x}$ finner man hur många sätt man kan med mängden *u* fördela jobbet bland alla lärare.

Pseudokod:

```
Ways(N, x, u[1,...,N])
⓪ if sum(u[0],..,u[N]) { < x , return 0
                       { = x , return 1
   A_{N,x+1} ← {∅}{∅}   #initiera tom matris
① for i←1 to x+1
      A_{1,i} ← { 1 , if u[1] >= i
                { 0 , else
② for i ← 1 to N do A_{i,1} ← 1
③ for i ← 2 to N do
      for j ← 2 to x+1
          A_{i,j} ← { Sum(A_{i-1,j} ,...., A_{i-1,1}), if u[i] >= j
                    { Sum(A_{i-1,j-0} ,...., A_{i-1,j-u[i]}), else
④ return A_{N,x+1}
```

Sum(): returnerar summan av elementen i en given array som körs på linjär tid för array storleken.

Mehir Seyum Wolde
CINTE3

0. Basfall som kontrollerar ifall summan av arrayen är exakt lika med x eller mindre.
1. Funktion för att initiera matrisens konstanter på första raden. Första raden fylls med 1:or tills respektive tal inte är större än möjliga x-värden.
2. Första kolumnen fylls med 1:or eftersom det endast finns 1 sätt att bilda 0 med alla tal.
3. Fyller resten av matrisen med rätt värde genom att använda tidigare beräknade värden för att få fram nya möjliga kombinationer.
4, Returnerar antalet sätt att utföra *x* uppgifter med *N* lärare där varje lärare kan göra upp till ett visst antal uppgifter beskrivet av *u*.

Tidskomplexitet:
De två första looparna kommer ge en komplexitet på *O(N+x)* eftersom att dessa endast utför konstanta operationer, tilldelning av värden och sökning i matrisen (i detta fall är matrisen noll indexerad). I den nestade for loopen kommer den yttersta loopen att ge en komplexitet på *O(N-1) = O(N)* eftersom att den ska loopa igenom alla möjliga tal i arrayen *u* från andra positionen i arrayen. I den inre for loopen kommer det att köras från 1 hela vägen upp till x vilket ger en tidskomplexitet på *O(x)*. Eftersom att vi har en gräns på hur många uppgifter en lärare kan göra, 10, får vi även en begränsning på möjliga x. I detta fall kan ett acceptabelt x endast bli 10 gång antalet lärare(*N*) vilket gör att komplexiteten kan ändras till *O(10N) = O(N)*. For loopen längst in kommer ge en komplexitet på *O(N)* eftersom att denna allmänt sagt gör liknande iterationer till loopen innan fast baklänges. Detta ger en övergripande polynomisk komplexitet på *O(N+10N +10N + N*N*N) = O(N^3)*.

Motivering för korrekthet av algoritmen:
För att visa att algoritmen algoritmen stämmer måste vi visa den uppfyller kravet på att antal sätt att skapa x uppgifter med hjälp av u samt visa att inga av looparna inte terminerar. Detta kan visas genom att visa att detta stämmer för ett givet basfall, alltså när summan av alla element är $\leq x$, för algoritmen. Med hjälp av induktion kan man sedan anta att algoritmen stämmer under ett tillstånd i matrisen *k.* Sedan med hjälp av detta antagande ska man visa att algoritmen stämmer för varje tillstånd *k+1* och att summan som beräknas är på rätt element i arrayen *u[1,...,N]* samt att det är rätt delsumma av *x* som beräknas.

Mehir Seyum Wolde
CINTE3

2.2 Skriv funktion print_all_ways som skriver ut alla sätt som jobbet kan fördelas på
Algoritmbeskrivning:
Algoritmen fungerar genom att testa alla möjliga värden på *u[i]* upp till en begränsad längd (*s*), samt det högsta möjliga värdet för den platsen, av arrayen tills den hittar ett värde som är lika med x. Då ställer den om värdet på den senaste pekarens plats för att testa nya kombinationer med alternerande värden i hjälp arrayen a[].

```
Print_all_ways(N, x, u[1,..,N])
    a[N] ← ∅
    S ← 1

    for i ← 0 to S do
       if i < n do
          While a[i] < u[i]
             a[i] + 1
             if(sum(a[1], ..., a[N]) = X ) do
                Print(a)
             if i > 0 do i = 0
          a[i] ← 0
          S + 1
       else return
```

Sum(): samma funktion som används i 2.1 för att beräkna summan av en array
print(): skriver ut alla element i en given array enligt formatet angett i uppgiften (x1+x2+...xn) vilket går på linjär tid gentemot storleken på arrayen