DAKI Programmeeropdracht 4: De zagerij

(Deze versie is van 5 oktober 2022, 01:43)

Je moet deze programmeeropdracht **zelf** en **alleen** maken. Je mag zeker met anderen overleggen over je aanpak, maar code van anderen bekijken of overnemen of zelf code delen met anderen is uitdrukkelijk niet toegestaan. Je moet je programma schrijven in C#, en inleveren via DOMjudge.

1 Opdrachtbeschrijving

De DAKIBOT gaat nu eens de servomotoren uit de mouwen steken, en de actuatoren vies maken! Het wordt tijd voor écht werk... in een fabriek! Het betreft een zagerij die je hebt gevonden op het light web, en waar je de DAKIBOT wil inzetten om een planningsprobleempje op te lossen. De bedoeling is vierkante plankjes te gaan verkopen, maar deze moeten eerste gefabriekeerd worden door rechthoekige planken in stukken te zagen.

Stel je een rechthoekige plank met (verticale) hoogte h en (horizontale) breedte b voor, die op een zaagtafel ligt. Uit zo'n plank van h bij b dakimeters kunnen $h \times b$ vierkante plankjes worden gezaagd, door $(h-1)$ horizontale en $(b-1)$ verticale zaagbewegingen te maken. Vanwege de dikte en de hardheid van de plank op verschillende plekken hebben de zaagbewegingen verschillende *kosten*. Elke zaagbeweging gaat over de *gehele* lengte of breedte van de plank, maar de kosten tellen elke keer dat een aparte plank wordt doorgezaagd. Als je bijvoorbeeld begint met een horizontale zaagbeweging, snijdt een daarop volgende verticale zaagbeweging twee horizontale planken door, en deze zaagbeweging bestaat dus uit *twee* zaagsneden, met als kosten twee keer de kosten van de verticale zaagbeweging. Er zijn in het algemeen dus $(h-1)+(b-1)$ *zaagbewegingen*, en elk daarvan levert één of meerdere *zaagsneden* op. Je moet de kosten optellen over alle zaagsneden. Zie het voorbeeld in sectie 3 hieronder.

De opdracht is om de *laagste kosten* te bepalen van een zaagplan, waarmee een gegeven plank in vierkantjes van 1 bij 1 dakimeter kan worden gezaagd, als de kosten van de $(h+b-2)$ zaagbewegingen gegeven zijn. Horizontale en verticale zaagbewegingen kunnen in het zaagplan zo vaak als nodig en zonder extra kosten met elkaar worden afgewisseld.

2 Invoer en uitvoer

Zoals altijd kan na de relevante input op elke regel nog meer tekst staan, die altijd door tenminste één spatie wordt voorafgegaan. Op de eerste regel van de input staan drie getallen h, b en k, alledrie ≥ 1. De getallen h en b geven de hoogte en breedte van de plank, beide $\leq 1\,000\,000$. De betekenis van de parameter k wordt in de sectie Algoritmische eisen uitgelegd. Na de eerste regel volgen $h-1$ regels met de kosten van de horizontale zaagbewegingen, en

$b-1$ regels met de kosten van de verticale zaagbewegingen. De uitvoer bestaat uit één regel met daarop twee getallen, gescheiden door een spatie. Het eerste getal is de kosten van het goedkoopste zaagplan om de gegeven plank in vierkantjes te zagen. Het tweede getal is het aantal zaagsneden in dat plan.

3 Voorbeeld

Bij de volgende input:

```
4 5 1 // de hoogte is 4, de breedte is 5
4  De hoogte is vier, dus        (h1)
1    er zijn drie                (h2)
2    horizontale zaagbewegingen. (h3)
4  De breedte is vijf,           (v1)
6    dus er zijn vier            (v2)
3    verticale.                  (v4)
1    zaagbewegingen.             (v5)
```

hoort de volgende output:

```
44 19
```

Zorg dat je goed begrijpt hoe dit antwoord tot stand komt. Hiervoor moet je al nadenken over een manier om het goedkoopste zaagplan op te stellen.

4 Algoritmische eisen

Je zult om een zaagplan te construeren, een ordening moeten aanbrengen in bepaalde objecten. Je moet voor die objecten een klasse schrijven die de `IComparable` interface implementeert. Verder moet je een algoritme schrijven dat een array van `IComparable` objecten sorteert. Je sorteeralgoritme moet dus werken op álle arrays met `IComparable` objecten, niet alleen degene in deze specifieke applicatie.

Je moet zelf het beschreven sorteeralgoritme implementeren, en je mag dus geen ingebouwde algoritmen voor sorteren gebruiken. Het algoritme moet een hybride zijn van een recursief en een iteratief algoritme. Het recursieve algoritme is *quicksort*, en het iteratieve algoritme is *selection sort*. Dergelijke hybride algoritmen komen in de praktijk veel voor (bijvoorbeeld in de C# `Array.Sort` methode (die een hybride van insertion sort, heapsort en quicksort implementeert), omdat de recursie teveel overhead oplevert bij kleine instanties, en een kwadratisch algoritme in die situaties sneller is. Dus élke keer dat je een arraysegment moet sorteren met een lengte kleiner dan of gelijk aan k (gespecificeerd in de input), moet je selection sort gebruiken.

5 Hints, tips

Geen.