Tentamen Algoritmen en Programma's (I00008)

dinsdag 16 januari 2007, 13:30 - 15:30 uur, HG00.303

- Dit tentamen is gesloten boek. Je mag geen literatuur of PC gebruiken.
- Deze toets bestaat uit twee opgaven, die beide even zwaar meetellen voor het cijfer. De waardering per onderdeel staat vermeld in de kantlijn.
- Schroom niet vragen te stellen over eventuele onduidelijkheden in de opgaven.
- Ga niet overhaast te werk en lees de opgaven nauwkeurig. Begin met de opgaven die je het eenvoudigst lijken. Denk eerst goed na. Hou je antwoorden kort en controleer ze voor je het werk inlevert.

Succes!

1 Sorteren: 'n verbeterde bubblesort

De fundamentele reden voor de slechte complexiteit van simpele sorteeralgoritmen als bubblesort en insertionsort is dat elementen die ver van hun juiste plek staan er te lang over doen om op de goede plek te komen. In deze opgave concentreren we ons op bubblesort. In bubblesort verwisselen we steeds buren totdat de elementen in de juiste volgorde staan. Na één keer door de lijst lopen ('n zogenaamde *sweep*) staat het grootste element wel achteraan, maar een klein element dat te ver naar achteren staat is slechts één plaats naar voren gekomen. Zo'n klein element dat helemaal verkeerd staat wordt om die reden wel een schildpad genoemd. De grote elementen heten dan hazen: die gaan snel naar de juiste plek. Om zo'n klein element n plaatsen naar voren te krijgen hebben we n sweeps door de rij nodig.

In Combsort heeft men de volgende oplossing voor dit probleem: in plaats van directe buren (`rij[i]` en `rij[i+1]`), vergelijken we elementen die verder, op afstand `gap`, uit elkaar liggen: `rij[i]` en `rij[i+gap]`. Als deze elementen niet in de juiste volgorde staan worden ze verwisseld (als in gewoon bubblesort). Na één keer op deze manier door de rij te lopen wordt de `gap` verkleind met een *shrink factor*. Zowel experimenteel als theoretisch heeft met vastgesteld dat een shrink factor van 1.3 (nagenoeg) optimaal is. Als de `gap` op een bepaald moment 20 is dan zal de volgende waarde $20/1.3 = 15.3846$ zijn. Bij conversie van een **`double`** naar **`int`** in C++ wordt het gedeelte achter de komma weggegooid. We krijgen altijd de eerste integer die kleiner is, in dit geval dus 15. Dat is precies wat je hier nodig hebt. Pas op: als de `gap` 1 is, zal die 0 worden bij de eerstvolgende aanpassing, tenminste, als je geen maatregelen neemt ($1/1.3 = 0.7692$ wat geconverteerd zal worden naar 0).

Initieel is de `gap` gelijk aan de lengte van de te sorteren rij gedeeld door de shrink factor. Zodra `gap` de waarde 1 heeft gekregen hebben we in feite een gewone bubblesort. In dat geval blijven we elementen verwisselen totdat de rij gesorteerd is, uiteraard zonder de `gap` verder te verkleinen. Omdat de elementen bijna op de goede plek staan, is hier altijd een klein aantal sweeps door de rij voldoende. Je mag in de analyse aannemen dat er maximaal C (een constante onafhankelijk van de lengte van de rij) sweeps met een `gap` van grootte 1 nodig zijn.

Afhankelijk van de lengte van de te sorteren rij komt het algoritme na enige tijd altijd bij een `gap` van 9, 10 of 11 uit (je kunt eenvoudig nagaan dat alle grotere waarden hierbij uitkomen, b.v. $12/1.3 = 9.2308$ en afgerond dus 9). Vanaf dat moment zijn er drie mogelijke reeksen van afnemende waarden voor de `gap`: $[9, 6, 4, 3, 2, 1]$, $[10, 7, 5, 3, 2, 1]$ en $[11, 8, 6, 4, 3, 2, 1]$. Het blijkt dat alleen de laatste reeks alle schildpadden doodt (zet ook de kleine elementen snel ongeveer

op de juiste plek) voordat de `gap` de waarde 1 heeft. In combsort11 zorgen we dat we altijd in dit gunstige geval terecht komen door de `gap` op 11 te zetten als die ooit 9 of 10 wordt.

10 punten **1.a**) Implementeer een functie **`void`** `combsort11(`**`int`** `rij [], `**`int`** `len)` die de boven beschreven variant van bubblesort implementeert.

Hint: Als de `gap` een integer is kun je (dankzij de automatisch type conversies) gewoon schrijven `gap = gap/shrink` om te gap te verkleinen, waarbij `shrink` als een **`double`**-constante met waarde 1.3 gedeclareerd is.

2 punten **1.b**) Hoe vaak zal er in `combsort11` door de rij gelopen worden als de rij al gesorteerd is? Wat is dus de best case complexiteit? Geef een korte motivatie!

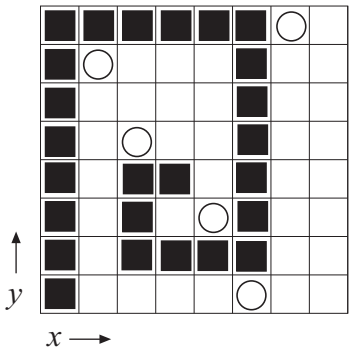
2 punten **1.c**) Hoe vaak moet er in het slechtste geval door de rij gelopen worden om de rij gesorteerd te krijgen? Wat is dus de worst case complexiteit? Geef een korte motivatie!

1 punten **1.d**) Wat is de gemiddelde complexiteit? Geef een korte motivatie!

2 Backtracking: spiralen

Hoewel het nog vrij kort geleden is dat hij zijn verjaardag vierde is Sinterklaas al weer volop bezig met de voorbereidingen voor het nieuwe jaar. Tijdens een van zijn nachtelijke ritten op zijn schimmel zag de Sint een laaghangende tak van een boom waar hij onderdoor reed over het hoofd en stootte met zijn staf tegen dit obstakel waardoor de krul afbrak. Uiteraard moet deze weer hersteld zijn voordat de feestelijkheden rond half november weer kunnen starten. Om die reden heeft de Sint bij de plaatselijke houthandel een vierkant stuk triplex aangeschaft waaruit hij de nieuwe krul gaat zagen. Als voorbereiding heeft hij het stuk hout opgedeeld in kleine vakjes die hij eerst inkleurt om daarna met zagen te kunnen beginnen. Wat het geheel nogal complex maakt is het feit dat het gekochte hout van slechte kwaliteit is: er zitten op diverse plaatsen gaten in die natuurlijk geen deel mogen uitmaken van de nieuwe krul. En Sinterklaas wil dat de nieuwe krul zo lang mogelijk wordt.

Jij schiet de goedheiligman te hulp door een programma te schrijven dat voor een gegeven plaat (met gaten) bepaalt wat de maximale lengte van de krul is. In onderstaand plaatje is 'n mogelijke krul (spiraal) aangegeven. Deze heeft echter *niet* de maximale lengte.



De opdracht luidt derhalve concreet: ontwerp en implementeer, uiteraard in C++, een *backtracking algoritme* waarmee voor een vierkant bestaande uit N^2 vakjes bepaald wordt hoeveel van deze vakjes maximaal deel uitmaken van de spiraal. Hierbij dien je rekening te houden met de volgende voorwaarden:

- De spiraal wordt van buiten naar binnen geconstrueerd beginnende linksonder met als startrichting noord. Dit startveld is niet gemarkeerd en kan dus altijd ingekleurd worden.
- De spiraal dient rechtsdraaiend te zijn dat wil zeggen dat bij de constructie van de spiraal vanaf iedere positie de tot dan toe gevormde spiraal op twee manieren kan worden uitgebreid, namelijk rechtdoor en rechtsaf.

- De spiraal mag zichzelf niet raken, d.w.z. ieder ingekleurd vakje heeft alleen met zijn directe voorganger en met zijn directe opvolger een zijde gemeen.
- Sommige van de vakjes zijn gemarkeerd als 'gat'. Deze gaten mogen geen deel uitmaken van de spiraal.

Om je 'n beetje op weg te helpen geven we enkele declaraties voor. Je mag hiervan gebruik maken in jouw uitwerking. We beginnen met wat eenvoudige types en een constante.

```
enum Richting { Noord, Oost, Zuid, West };

Richting Rechtsaf (Richting r)
{   return Richting ((r+1) % 4); }

enum Vak { Leeg, Gat, Krul }; // De representatie van de vakjes
                              // waaruit de plaat is opgebouwd
const int Grootte = 8; // De grootte van de triplex plaat.
```

De administratie die nodig is voor het eigenlijke algoritme stoppen we in een klasse:

```
class Spiraal
{
private:
    Vak plaat [Grootte+2][Grootte+2];

    int verleng (Richting r, int x, int y);

public:
    int langste (Richting r=Noord, int x=1, int y=1);

    void toon (); // Toont de plaat.
    Spiraal ();   // De constructor die een ongekleurde plaat
                  // maar wel met gaten oplevert.

} ;
```

Om te voorkomen dat je tijdens het uitbreiden van de spiraal mogelijk buiten de plaat terecht komt mag je aannemen dat deze wordt omgeven door vakjes waarin een gat zit. Dit verklaart tevens waarom de plaat in beide dimensies 2 vakjes groter is geworden.

De publieke methode genaamd `langste` bepaalt de langst mogelijke spiraal voor de gegeven plaat. Hierin kan de hulpmethode `verleng` worden aangeroepen die de gegeven spiraal in de aangegeven richting probeert uit te breiden. De definitie geven we voor.

```
int Spiraal :: verleng (Richting r, int x, int y)
{
    switch (r)
    {   case Noord: return langste (r, x, y+1);
        case Oost:  return langste (r, x+1, y);
        case Zuid:  return langste (r, x, y-1);
        case West:  return langste (r, x-1, y);
    }
}
```

De `Spiraal`-klasse kan naar believen worden uitgebreid met methodes en/of variabelen.

11 punten **2.a**) Implementeer de methode `langste` die de lengte van de langst mogelijke spiraal bepaalt.

4 punten **2.b**) Geef duidelijk aan hoe je bovenstaande klasse moet wijzigen/uitbreiden om niet alleen de lengte van de gevonden spiraal maar ook de spiraal zelf laat zien.