Aufgabe 4: Kleingärten

Team: Jan Niklas Groeneveld
Team-ID: 00828

22. November 2018

Inhaltsverzeichnis

1 Lösungsidee

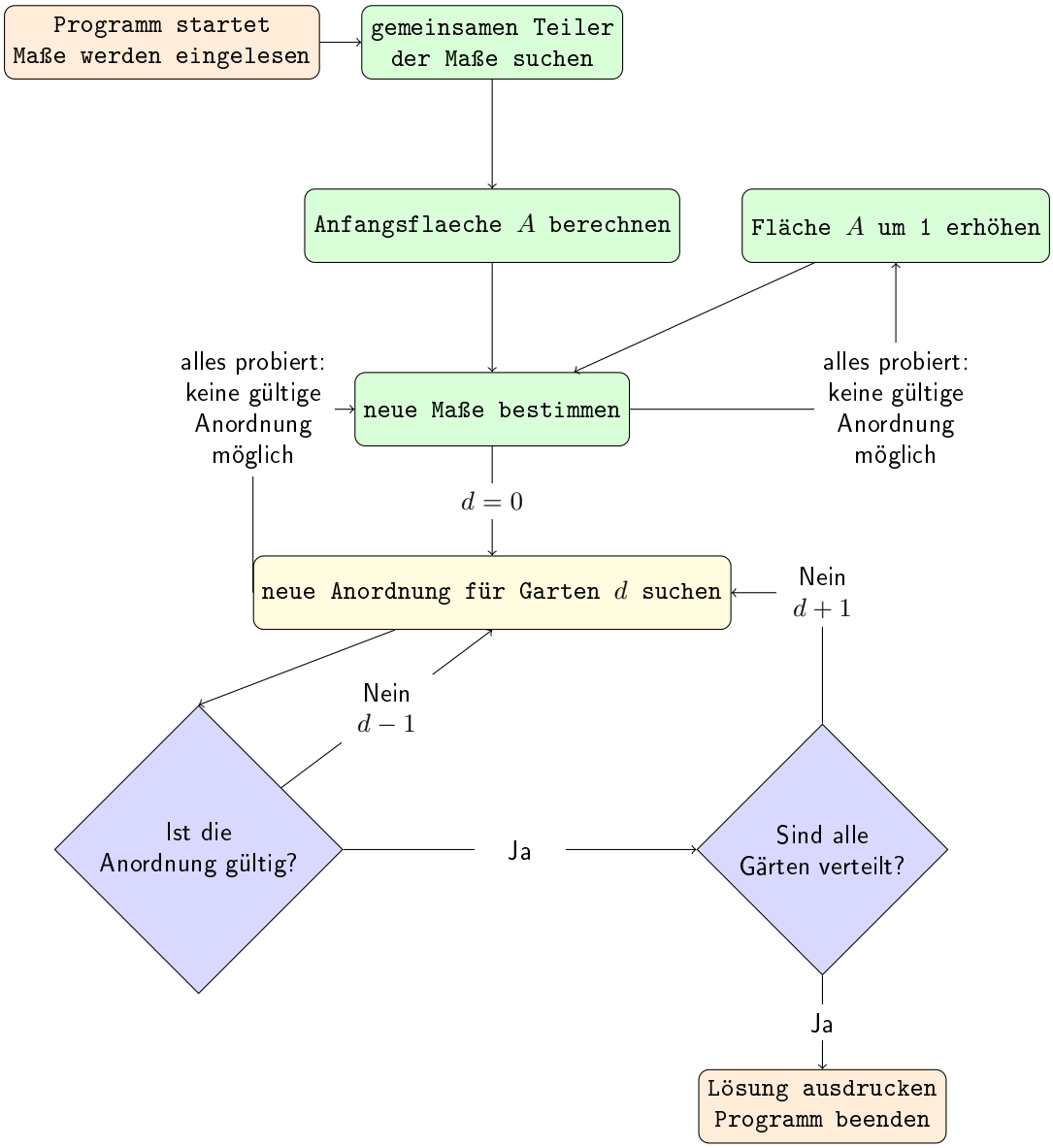
Ziel der Aufgabe ist es, eine vorgegebene Menge an Kleingärten, die in Breite und Höhe vorgegeben sind, auf einer rechteckigen Fläche so zu verteilen, dass diese Fläche möglichst klein ist. Dazu kann zuerst geprüft werden, ob die Maße der Gärten einen gemeinsamen Teiler besitzen, wie in dem Beispiel der Aufgabenstellung. Dort sind alle Maße ein Vielfaches der Fünf. In diesem Fall können die Maße durch diesen gemeinsamen Teiler dividiert werden, um später den Rechenaufwand gering zu halten. Da jeder Garten eine eigene Fläche hat, die durch das Produkt aus Breite und Höhe vorgegeben wird, lässt sich durch Addition dieser Einzelflächen das theoretische Minimum errechnen, das, falls dies möglich ist, erreicht werden soll. Um die Gärten auf dieser Fläche verteilen zu können, was später genauer erläutert wird, müssen zunächst ihre Maße errechnet werden. Dazu werden auf folgende Art ganzzahlige Teiler der Fläche gesucht, mit denen ein ebenfalls ganzzahliger Quotient ermittelt werden soll. Ist auf diese Art eine x- und y-Abmessung für die Fläche gefunden, wird die Anordnung der einzelnen Gärten auf folgende Art ausprobiert: Der erste Garten wird auf der Fläche zu erst in der linken oberen Ecke angeordnet. Dann folgen die nächsten Gärten mit ebenfalls aufsteigenden x- und y- Werten. Sollte die Fläche eines Gartens sich entweder mit der eines anderen Gartens überschneiden oder die Grenzen des nach den oben erklärten Gebietes verletzen, ist die Anordnung ungültig und eine neue Position muss gesucht werden. Dabei wird es als sinnvoll erachtet, in bestimmten Abschnitten auch die Ausrichtung des Gartens durch 90°-Drehung zu ändern, um ein möglichst allgemein anwendbares Programm zu erreichen. Ist für einen Garten keine gültige Anordnung auf dem Gebiet zu erreichen, muss für die Anordnung des vorher gesetzten Gartens ebenfalls eine neue Anordnung gesucht werden. Wird eine Anordnung erreicht, bei der für alle Gärten eine gültige Anordnung möglich ist, wird diese ausgegeben und das Programm beendet. Stellt sich hingegen heraus, dass für das Gebiet der festgelegten Maße keine mögliche Anordnung erreichbar ist, muss die Größe der Fläche, in dem die Gärten angeordnet werden, erhöht werden. Das geschieht in Einserschritten mit anschließender erneuter Teilersuche und der erneuten Anordnung der Gärten.

Durch dieses Vorgehen erreicht der Algorithmus folgende Vorteile:

1. Es wird immer die bestmögliche Lösung erreicht.
2. Das Feld, in dem die Anordnungen der Gärten probiert werden, ist nie größer als absolut notwendig, da ein größeres Feld mehr Kombinationsmöglichkeiten bedeutet, die einen höheren Zeitaufwand zur Folge haben.

3. Die erste erreichte Lösung ist auf jeden Fall die beste. Nach Erreichen dieser Lösung kann das Programm sofort beendet werden.

Hier ist der Algorithmus als Flussdiagramm grafisch dargestellt:



2 Umsetzung

Die Maße für die Grundstücke werden in einer Zeigerstruktur gespeichert. Diese hat die Ganzzahlen für x- und y-Längen sowie die Fläche des Grundstückes. Die Zeiger auf diese Strukturen werden nach dem Einlesen der Maße und der Berechnung der Flächen in einem geeigneten Array gespeichert. Dann beginnt das Suchen nach einem gemeinsamen Teiler auf folgende Art: Ein Array von ausreichender Länge, der im abgegebenen Code auf 100 begrenzt ist, wird initialisiert, indem jeder seiner Werte auf null gesetzt wird. Dann wird eine dreifach verschachtelte Zählschleife genutzt. Die erste dieser drei Schleifen zählt, solange die Zählvariable kleiner ist als die Anzahl der Grundstücke, um jedes Grundstück zu berücksichtigen. Die zweite zählt nur von null bis eins, was für die x- und y- Ausdehnung genutzt wird, und die letzte zählt von null bis zum Wert der Abmessung. Wenn die Modulooperation des Wertes der jeweiligen Abmessung und der Zählvariable der inneren Zählschleife null ergibt, wird im eingangs initialisierten Array das Element des Wertes der inneren Zählschleife um eins erhöht. Zum Schluss zählt eine Zählschleife von null bis unter jenen Maximalwert, der in dem Programm 100 beträgt. Falls der Inhalt des Arrays an dieser Stelle gleich der doppelten Anzahl der Grundstücke ist, bedeutet das, dass sowohl die

x-, als auch die y-Ausdehnung eines jeden Grundstückes durch diesen Wert teilbar ist. Die Ganzzahlvariable `groessterteiler` wird dann auf diesen Wert gesetzt, sodass der größte gemeinsame Teiler in dieser Variable gespeichert ist. Falls diese `teilersuchen()`-Funktion eine Zahl ungleich eins zurückgibt, wird für jedes Grundstück die Funktion `teilen()` aufgerufen. Sie dividiert die Längen und errechnet auch die Grundstücksfläche neu. Anschließend wird der zweidimensionale Array initialisiert, der für das spätere Ausprobieren der Kombinationen notwendig ist. Dieser besitzt eine maximale Kantenlänge von 100x100, die mit den Beispielen nicht ausgeschöpft werden, aber dennoch in Gänze mit -1 aufgefüllt werden, sodass ein problemloser Betrieb möglich ist, weil -1 für einen ungenutztes Feld steht. Nachdem die Funktion `calc_total_area()` aufgerufen wurde, die mit einer Zählschleife durch den Array der Zeiger auf die Strukturen, in denen die Grundstücksdaten gespeichert sind, durchgeht, dabei die Einzelflächen zusammenaddiert und anschließend zurückgibt, wird die Funktion `flaechen_ausprobieren()` aufgerufen. Sie arbeitet mit einer verschachtelten Zählschleife. Die erste dieser beiden Schleifen zählt ab dem Wert der Gesamtfläche aufwärts. Das implementiert den Aspekt der Grundidee, dass von der minimal benötigten Fläche in Einserschritten hochgezählt wird, wenn das notwendig wird. Die zweite Zählschleife beginnt mit der Wurzel der auszuprobierenden Fläche und zählt herunter, solange der Wert größer ist als null. In einer Abfrage wird im Anschluss geprüft, ob die Modulooperation der auszuprobierenden Fläche mit der Zählvariable der inneren Schleife null ergibt. Sollte das der Fall sein, bilden diese Zählvariable und der Quotient aus auszuprobierender Fläche und der Zählvariable ein Wertepaar, das als Kantenlängen für das Feld, in dem ausprobiert wird, genutzt wird. Die dann aufzurufende Funktion `ausprobieren()`, die einen boolschen Wert zurückgibt, bildet das Zentralelement des Algorithmus. Sie wird unten erläutert. Sollte dieser Rückgabewert `true` sein, wurde bei dieser Fläche eine gültige Lösung erreicht. Jetzt werden die Kantenlängen, die Gesamtfläche und die Effizienz der Kombination, die sich als Quotient der Minimalfläche und der ausprobierten Fläche ergibt. Anschließend wird die Funktion verlassen. Sollte `false` zurückgegeben werden, wird mit dem nächsten Schleifendurchläufen fortgesetzt, also das nächste Wertepaar ausprobiert, oder, falls die innere Zählschleife durchgelaufen ist, die Fläche um eins erhöht.

Die `ausprobieren(int depth)`-Funktion arbeitet mit einer verschachtelten Zählschleife. Hier wird einmal von null bis zur x-Ausdehnung des Feldes, und einmal zur y-Ausdehnung gezählt. Eine dritte, darin enthaltene Zählschleife zählt von null bis eins. Wenn dieser Wert eins ist, wird das Feld um 90° gedreht, also die x- und y- Werte werden getauscht. Anschließend wird die Funktion `reserve(int depth, int x, int y)` aufgerufen. Sie gibt `true` zurück, falls das Grundstück mit der Nummer `depth` mit der jeweiligen Ausrichtung an der x- und y-Koordinate der verschachtelten Zählschleife gültig positionierbar ist. Wenn dies der Fall ist, wurde in der Funktion dieser Platz reserviert. Die genaue Funktionsweise der Funktion wird im Anschluss erläutert. Wenn der momentane Wert von `depth` mit eins addiert noch kleiner ist als die Anzahl der Grundstücke, wird die Funktion `ausprobieren(int depth)` unter Übergabe des um eins erhöhten momentanen Wertes von `depth` rekursiv aufgerufen. Sollte hierbei `true` zurückgegeben werden, wird das momentane Feld wieder sauber von der Fläche gelöscht, und ebenfalls `true` zurückgegeben. Sollte der mit eins addierte Wert von `depth` nicht kleiner sein als die Anzahl der Grundstücke, ist er im Kontext dieses Programmes gleich der Anzahl der Grundstücke. Hier ist also eine gültige Anordnung aller Gärten möglich gewesen. Die Fläche wird so ausgedruckt, dass auf der Kommandozeile die Anordnung der Gärten erkennbar ist, der momentane Garten wird entfernt und `true` zurückgegeben. Auf jeden Fall wird durch die Funktion `reset(int depth, int x, int y)` das jeweilige Feld gelöscht, bevor es zu einem erneuten Schleifenaufruf kommt. Am Ende der Funktion wird `false` zurückgegeben. Diese Zeile wird nur erreicht, wenn bei keiner Anordnung `true` zurückgegeben wurde.

Die Funktion `reserve(int depth, int x, int y)`, die zum reservieren des Platzes eines Gartens auf der Fläche verwendet wird, prüft als erstes, ob entweder die x- oder y-Position des zu reservierenden Stückes plus die x- bzw. y-Ausdehnung des Gartens die x- bzw. y- Ausdehnung der für die Anordnung zu verwendenden Fläche überschreitet. In diesem Fall wird sofort `false` zurückgegeben. Als nächstes zählt eine verschachtelte Zählschleife bis zur x- bzw. y-Ausdehnung des jeweiligen Gartens. Die Summe der x- bzw. y- Position und der jeweiligen Zählvariable befindet sich so immer im zu reservierenden Gebiet. Sollte an einer dieser Stellen der Wert des zweidimensionalen Arrays ungleich -1 sein, ist es bereits belegt. In diesem Fall muss auch `false` zurückgegeben werden. Wenn das nicht der Fall ist, füllt eine gleich aufgebaute verschachtelte Zählschleife den Bereich mit dem jeweiligen Wert von `depth` auf, sodass das Feld für den Garten mit dieser Nummer reserviert ist.

3 Beispiele

Dies ist das erste Beispiel:

```
Wie viele Gaerten sollen angelegt weden? (maximal 10)
4
Welche Abmessungen soll der naechste Garten mit der Nummer 0 haben? (x*y)
15*25
Welche Abmessungen soll der naechste Garten mit der Nummer 1 haben? (x*y)
30*15
Welche Abmessungen soll der naechste Garten mit der Nummer 2 haben? (x*y)
25*15
Welche Abmessungen soll der naechste Garten mit der Nummer 3 haben? (x*y)
20*25
Man kann diese Gaertenlaengen durch 5 dividieren, damit weniger zu rechnen ist.
Der Ausdruck ist um diesen Maßstabsfaktor verkleinert.
Bitte beachten Sie, dass sich die Angaben ueber die Flaeche zum Quadrat dieses
Faktors verkleinern.
0 0 0 0 0 2 2 2
0 0 0 0 0 2 2 2
0 0 0 0 0 2 2 2
1 1 1 . . 2 2 2
1 1 1 . . 2 2 2
1 1 1 3 3 3 3 3
1 1 1 3 3 3 3 3
1 1 1 3 3 3 3 3
1 1 1 3 3 3 3 3
Diese Loesung hat eine Flaeche von 72 bei Kantenlaengen von 8 und 9. Das
bedeutet eine Effizienz von 94.44%, weil die minimale Flaeche 68 gewesen waere.
Das Programm ist beendet.
```

Das zweite Beispiel beinhaltet fünf Gärten:

```
Wie viele Gaerten sollen angelegt weden? (maximal 10)
5
Welche Abmessungen soll der naechste Garten mit der Nummer 0 haben? (x*y)
6*3
Welche Abmessungen soll der naechste Garten mit der Nummer 1 haben? (x*y)
2*2
Welche Abmessungen soll der naechste Garten mit der Nummer 2 haben? (x*y)
3*1
Welche Abmessungen soll der naechste Garten mit der Nummer 3 haben? (x*y)
4*4
Welche Abmessungen soll der naechste Garten mit der Nummer 4 haben? (x*y)
4*4
0 0 0 2 2 2 .
0 0 0 3 3 3 3
0 0 0 3 3 3 3
0 0 0 3 3 3 3
0 0 0 3 3 3 3
0 0 0 4 4 4 4
1 1 . 4 4 4 4
1 1 . 4 4 4 4
. . . 4 4 4 4
Diese Loesung hat eine Flaeche von 63 bei Kantenlaengen von 7 und 9. Das
bedeutet eine Effizienz von 90.48%, weil die minimale Flaeche 57 gewesen waere.
Das Programm ist beendet.
```

Im letzten Beispiel der BWINF-Seite werden ebenfalls fünf Gärten kombiniert:

```
Wie viele Gaerten sollen angelegt weden? (maximal 10)
5
Welche Abmessungen soll der naechste Garten mit der Nummer 0 haben? (x*y)
4*4
Welche Abmessungen soll der naechste Garten mit der Nummer 1 haben? (x*y)
2*3
Welche Abmessungen soll der naechste Garten mit der Nummer 2 haben? (x*y)
6*1
Welche Abmessungen soll der naechste Garten mit der Nummer 3 haben? (x*y)
5*2
Welche Abmessungen soll der naechste Garten mit der Nummer 4 haben? (x*y)
3*5
0 0 0 0 1 1
0 0 0 0 1 1
0 0 0 0 1 1
0 0 0 0 . 2
3 3 4 4 4 2
3 3 4 4 4 2
3 3 4 4 4 2
3 3 4 4 4 2
3 3 4 4 4 2
Diese Loesung hat eine Flaeche von 54 bei Kantenlaengen von 6 und 9. Das
bedeutet eine Effizienz von 98.15%, weil die minimale Flaeche 53 gewesen waere.
Das Programm ist beendet.
```

Die Effizienz des Algorithmus lässt sich besonders gut dann verdeutlichen, wenn die Zahl der Gärten erhöht wird. Hier ist ein Beispiel mit zehn Gärten, das in sehr kurzer Zeit gerechnet wurde:

```
Wie viele Gaerten sollen angelegt weden? (maximal 10) 10
Welche Abmessungen soll der naechste Garten mit der Nummer 0 haben? (x*y) 1*3
Welche Abmessungen soll der naechste Garten mit der Nummer 1 haben? (x*y) 2*4
Welche Abmessungen soll der naechste Garten mit der Nummer 2 haben? (x*y) 3*2
Welche Abmessungen soll der naechste Garten mit der Nummer 3 haben? (x*y) 4*4
Welche Abmessungen soll der naechste Garten mit der Nummer 4 haben? (x*y) 5*2
Welche Abmessungen soll der naechste Garten mit der Nummer 5 haben? (x*y) 6*3
Welche Abmessungen soll der naechste Garten mit der Nummer 6 haben? (x*y) 2*2
Welche Abmessungen soll der naechste Garten mit der Nummer 7 haben? (x*y) 3*4
Welche Abmessungen soll der naechste Garten mit der Nummer 8 haben? (x*y) 4*1
Welche Abmessungen soll der naechste Garten mit der Nummer 9 haben? (x*y) 5*1
0 1 1 2 2 2 6 6
0 1 1 2 2 2 6 6
0 1 1 4 4 4 4 4
8 1 1 4 4 4 4 4
8 5 5 5 5 5 5 9
8 5 5 5 5 5 5 9
8 5 5 5 5 5 5 9
3 3 3 3 7 7 7 9
3 3 3 3 7 7 7 9
3 3 3 3 7 7 7 .
3 3 3 3 7 7 7 .
Diese Loesung hat eine Flaeche von 88 bei Kantenlaengen von 8 und 11. Das
bedeutet eine Effizienz von 97.73%, weil die minimale Flaeche 86 gewesen waere.
Das Programm ist beendet.
```

Hier ist ein zusätzliches Beispiel, bei dem eine Flächennutzung von 100% erreicht wird:

```
Wie viele Gaerten sollen angelegt weden? (maximal 10) 4
Welche Abmessungen soll der naechste Garten mit der Nummer 0 haben? (x*y)
4*3
```

```
Welche Abmessungen soll der naechste Garten mit der Nummer 1 haben? (x*y)
2*6
Welche Abmessungen soll der naechste Garten mit der Nummer 2 haben? (x*y)
3*3
Welche Abmessungen soll der naechste Garten mit der Nummer 3 haben? (x*y)
2*1
0 0 0 3 3
0 0 0 1 1
0 0 0 1 1
0 0 0 1 1
2 2 2 1 1
2 2 2 1 1
2 2 2 1 1
Diese Loesung hat eine Flaeche von 35 bei Kantenlaengen von 5 und 7. Das
bedeutet eine Effizienz von 100.00%, weil die minimale Flaeche 35 gewesen waere.
Das Programm ist beendet.
```

4 Quellcode

Die Funktion `main()`:

```
int main()
{
  // Anzahl und Abmessungen der Gaerten werden erfragt
  printf("Wie viele Gaerten sollen angelegt weden? (maximal %d)\n", MAXGRUND);
  scanf("%d", &anzahlgrund);
  for (int i = 0; i < anzahlgrund; i++)
  {
    printf("Welche Abmessungen soll der naechste Garten mit der Nummer %d haben? (x*y)\n"
    , i);
    int x = 0;
    int y = 0;
    scanf("%d*%d", &x, &y);
    liste[i] = new_stueck(x, y);
  }
  int groessterteiler = teilersuchen(); // Groessten gemeinsamen Teiler suchen, teilen
    und Meldung ausgeben
  if (groessterteiler != 1)
  {
    printf("Man kann diese Gaertenlaengen durch %d dividieren, damit weniger zu rechnen
    ist.\nDer Ausdruck ist um diesen Maßstabsfaktor verkleinert.\nBitte beachten Sie,
    dass sich die Angaben ueber die Flaeche zum Quadrat dieses Faktors verkleinern.\n",
    groessterteiler);
    teilen(groessterteiler);
  }
  init_flaeche(); // globaler 2D-Array der fuer das Ausprobieren wichtigen Flaeche wird
    so initialisiert, dass ueberall -1, also frei eingetragen wird.
  calc_total_area(); // Gesamtflaeche wird als Summe der Einzelflaechen berechnet und in
    die globale Variable "gesamtflaeche" geschrieben.
  bool test = flaechen_ausprobieren(); // fuer Funktionsweise: siehe Funktion
  printf("%s", (test == true) ? "Das Programm ist beendet.\n" : "Es ist ein Fehler
    aufgetreten");
  return 0;
}
```

Für das gesamte Programm wird die Struktur `Wunschstueck` benötigt. So ist sie programmiert:

```
typedef struct Wunschstueck
{
    int x;
    int y;
    int grundstuecksflaeche;
} Wunschstueck;
```

Wie man am ersten Beispiel der BWINF-Seite sehen kann, ist es sehr effizient, wenn man für die Längen einen gemeinsamen Teiler sucht. Hier ist die Funktion `teilersuchen()`:

```
int teilersuchen()
{
  int teiler[MAXKANTE]; // In diesem Array wird die Anzahl der Grundstuecksausmasse
    gefuehrt, die durch den Wert des jeweiligen Elements teilbar sind.
  for (int i = 0; i < MAXKANTE; i++)
    teiler[i] = 0;
  for (int i = 0; i < anzahlgrund; i++) // Die aeussere Zaehlschleife prueft zaehlt alle
    Grundstuecke durch.
  {
    for (int j = 0; j < 2; j++)                          // Diese Zaehlschleife zaehlt
    nur von 0 bis 1.
    {
      int test = (j == 0) ? liste[i]->x : liste[i]->y;  // Mit den Werten aus dieser
    mittleren Zaehlschleife wird bestimmt, ob die x- oder y-Variable auf Teilbarkeit
    geprueft wird.
      for (int k = 2; k <= test; k++) // Alle Zahlen ab zwei bis zum Wert werden auf
    Teilbarkeit geprueft
      {
        if (test % k == 0)
          teiler[k]++;
      }
```

```
    }
  }
  int groesssterteiler = 1;
  for (int i = 2; i < MAXKANTE; i++)
  {
    if (teiler[i] == anzahlgrund * 2)
      groesssterteiler = i;
  }
  return groesssterteiler;
}
```

Um die richtigen Wertepaare zu finden, wird die Funktion `flaechen_ausprobieren()` genutzt. Hier ist sie abgedruckt:

```
bool flaechen_ausprobieren()
{
  for (int i = gesamtflaeche; i < (MAXKANTE * MAXKANTE); i++) // Diese aeussere
    Zaehlschleife erhoeht die Flaeche immer um 1, falls zuvor keine gueltige Anordnung
    erreicht werden konnte.
  {
    for (int j = (int)sqrt(i); j > 0; j--) // Diese innere Zaehlschleife beginnt bei der
    Wurzel der Flaeche, um Teilerpaare zu suchen.
    {
      if (i % j == 0) // Fuer die auszuprobierende Flaeche werden ganzzahlige Teiler
    benoetigt. Nur wenn die Division
      { // der Flaeche mit dem ersten Teiler keinen Rest uebrig laesst, ist das Wertepaar
     geeignet.
        xkante = j;
        ykante = i / j;
        if (ausprobieren(0) == true) // Mit dem Wertepaar fuer die Flaeche soll eine
    Kombination ausprobiert werden.
        { // Wenn eine Kombination moeglich ist, wurde in der Funktion bereits das
    Ergebnis ausgedruckt. Die Daten folgen hier:
          printf("Diese Loesung hat eine Flaeche von %d bei Kantenlaengen von %d und %d.
    Das bedeutet eine Effizienz von %3.2lf%c, weil die minimale Flaeche %d gewesen waere
    .\n", i, xkante, ykante, ((double)gesamtflaeche / (double)i) * 100, (char)37,
    gesamtflaeche);
          return true;
        }
      }
    }
  }
  return false;
}
```

Das eigentliche Herzstück des Algorithmus ist die Funktion `ausprobieren()`, die rekursiv die Kombinationsmöglichkeiten durchprobiert:

```
bool ausprobieren(int depth)
{
  for (int i = 0; i < xkante; i++) // Diese verschachtelte Zaehlschleife probiert alle
    denkbaren Positionen auf dem Feld aus.
  {
    for (int j = 0; j < ykante; j++)
    {
      for (int z = 0; z < 2; z++)
      {
        if (z == 1) // Anhand des Wertes der inneren Zaehlschleife wird bestimmt, ob das
    Grundstueck gedreht wird, um diese Moeglichkeiten auch zu beruecksichtigen.
        {
          int hilf = liste[depth]->x;
          liste[depth]->x = liste[depth]->y;
          liste[depth]->y = hilf;
        }
        if (reserve(depth, j, i) == true) // Der Platz fuer das Grundstueck wird
    reserviert.
        {
          if (depth + 1 < anzahlgrund) // Falls noch nicht alle Grundstuecke verteilt
    wurden, wird mit den Grundstueck der naechsten Stufe fortgefahren
          {
            if (ausprobieren(depth + 1) == true)
```

```
            {
              reset(depth, j, i);
              return true;
            }
          }
          else // Andernfalls wurden bereits alle Grundstuecke gueltig verteilt. Dann ist die Aufgabe geloest!
          {
            printflaeche();
            reset(depth, j, i);
            return true;
          }
          reset(depth, j, i); // Nach Abschluss des Durchlaufes muss die Reservierung wieder geloescht werden.
        }
      }
    }
  }
  return false;
}
```

Für das Reservieren des Platzes auf der Fläche ist die Funktion `reserve()` zuständig:

```
bool reserve(int depth, int x, int y) // Dise Funktion reserviert den Platz fuer ein Grundstueck
{
  if (x + liste[depth]->x > xkante || y + liste[depth]->y > ykante) // Wenn das Grundstueck den benutzbaren Bereich verlaesst, ist dieser Platz fuer diesen Garten ungueltig.
    return false;
  for (int i = 0; i < liste[depth]->y; i++)
  {
    for (int j = 0; j < liste[depth]->x; j++)
    {
      if (flaeche[x + j][y + i] != -1) // Wenn an einer Stelle, an der der Platz reserviert werden soll, bereits ein Garten geplant wird, ist der Platz ebenfalls ungueltig.
        return false;
    }
  }
  for (int i = 0; i < liste[depth]->y; i++)
  {
    for (int j = 0; j < liste[depth]->x; j++)
      flaeche[x + j][y + i] = depth; // Hier wird der Platz reserviert.
  }
  return true;
}
```