J1 Luftballons

Lösungsidee

In 10 Fächern einer Maschine befinden sich unterschiedliche Anzahlen von Luftballons. Nun soll man einige Fächer so auswählen, dass sich daraus insgesamt möglichst genau 20 Luftballons (und auf keinen Fall weniger als 20 Luftballons) entnehmen lassen. Diese Aufgabenstellung lässt sich in ein Grundproblem und ein paar einfachere Nebenprobleme zerlegen. Als Grundproblem betrachten wir folgende Aufgabenstellung:

Eingabe: Ein Array der Länge 10 mit positiven, ganzzahligen Werten.
Gesucht: Array-Elemente (und deren Positionen), die in der Summe mindestens 20 und möglichst genau 20 ergeben.

Bei näherem Nachdenken und wenn das Nachfüllen der Fächer mit berücksichtigt wird, ergeben sich außerdem die folgenden Fragen:

1. Wenn es zwei verschiedene Möglichkeiten gibt, die gleiche Summe zu erhalten, welche soll dann gewählt werden?
2. Wenn selbst bei Verwendung aller 10 Werte nur eine Summe kleiner 20 möglich ist, was ist dann zu tun?
3. Mit den aktuell vorhandenen Werten lässt sich 20 nicht exakt erreichen. Kann man mit den Ballons, die nachgefüllt werden, genau 20 erhalten?

Auf diese Fragen gute Antworten zu finden, stellt jeweils ein eigenes „Nebenproblem“ dar, das unabhängig vom Grundproblem betrachtet werden kann.

Grundproblem

Bei der Lösung des Grundproblems geht es darum, unter den vielen Möglichkeiten, die Array-Elemente miteinander zu kombinieren, die beste(n) auf geeignete Weise zu bestimmen. Dafür wollen wir zwei Strategien vorstellen, eine leicht zu findende und eine etwas komplexere.

Strategie 1: Brute Force Da es nur 10 Speicherfächer gibt, gibt es auch nur $2^{10} = 1024$ verschiedene Teilmengen von Fächern. Daher kann man ohne praktische Laufzeitprobleme für jede mögliche Teilmenge an Fächern die Anzahl an Ballons ausrechnen und dann die besten Möglichkeiten betrachten.

Strategie 2: Bottom-Up Idee: Wann lässt sich die Summe S durch Nutzung der ersten i Fächer erreichen?

- Wenn in Fach i genau S Ballons liegen.
- Wenn ich die Summe S durch Nutzung der ersten $i-1$ Fächer erhalten kann.
- Wenn ich die Summe $S - Fach[i]$ durch Nutzung der ersten $i-1$ Fächer erhalten kann.

Mit diesem Ansatz lässt sich eine 2-dimensionale Tabelle von Wahrheitswerten füllen: In dieser wird gespeichert, ob es mit den ersten i Fächern möglich ist, die Summe S zu erreichen.

Die Werte der Tabelle erhält man „Bottom-Up“, also indem man mit einfach zu berechnenden Werten beginnt und dann nach und nach weitere Werte aus schon bekannten Werten berechnet: Die Werte der ersten Spalte hängen von keiner anderen Spalte ab und werden zuerst berechnet. Die Werte der zweiten Spalte hängen nur von der ersten ab und können jetzt berechnet werden. Entsprechend wird für die anderen Spalten vorgegangen.

Falls man zu einem Zeitpunkt schon eine gültige Lösung gefunden hat, muss man alle schlechteren S nicht mehr betrachten.

Um am Ende die besten Möglichkeiten abzulesen, betrachtet man die Spalte für $i = 10$. Dort sucht man nach den Feldern $S \geq 20$, in denen `Wahr` steht. Die Fächer, die für diese Summe benötigt werden, kann man aus der Tabelle ablesen. Dazu muss man sich wiederholt fragen: *Warum wurde dieses Feld auf* `Wahr` *gesetzt?*

Nebenprobleme

Zur Lösung der Nebenprobleme lassen sich verschiedene Strategien überlegen. Die unten genannten Strategien liefern gute Ergebnisse, jedoch lassen sich immer Füllfolgen finden, für die sie nicht optimal funktionieren.

1. Wenn es zwei verschiedene Möglichkeiten gibt, die gleiche Summe zu erhalten, welche soll dann gewählt werden? Eine einfache Strategie ist es, die Kombination zu wählen, die die kleinere Anzahl an Fächern verwendet. Dadurch bleiben meist mehr Fächer mit kleineren Werten übrig. Mit diesen ist es später leichter, auf exakt 20 aufzufüllen. Mit größeren Werten schießt man leichter über die 20 hinaus.

2. Wenn selbst bei Verwendung aller 10 Elemente die Summe 20 nicht erreicht wird, was ist dann zu tun? Es empfiehlt sich, die kleinste Anzahl an Ballons auszukippen. Dadurch ist die Summe, die noch gebildet werden muss, am größten, und es gibt daher mehr Möglichkeiten, diese zu bilden.

3. Mit den aktuell vorhandenen Werten lässt sich 20 nicht exakt erreichen. Kann man mit den Ballons, die nachgefüllt werden, genau 20 erhalten? Gleiche Idee wie bei der zweiten Frage: Man nimmt die kleinste Anzahl an Ballons, die in der (im Moment) besten Summe vorkommt. Falls sich durch die nachgefüllten Ballons noch eine bessere Kombination ergibt, wird diese gewählt. Falls nicht, hat man trotzdem die bestmögliche Lösung.

Wer das in der Aufgabenstellung verlinkte Video zur LVM in der Sendung mit der Maus noch nicht gesehen hat, sollte das wirklich noch nachholen. Es ist zur Lösungsfindung nützlich und ruft Erinnerungen an einen jüngeren Ralph Caspers wach: `http://bit.ly/29PWZhx`

Implementierung

Bei der Implementierung gibt es keine besonderen Hürden. Die Datenstrukturen sind durch den gewählten Ansatz schon klar gegeben. Es sollte nur darauf geachtet werden, dass man nicht „schummelt“ und Kenntnisse über die Füllfolge mit in den Algorithmus einfließen lässt. Falls man eine objektorientierte Sprache verwendet, lässt sich das zum Beispiel realisieren, indem

man die LVM in eine eigene Klasse packt, und nur die aktuellen Füllstände nach außen sichtbar macht. Die Füllfolge lässt sich intern z.B. als Queue modellieren.

Beispiele

Die hier vorgestellte Lösung liefert für die sieben Beispiele insgesamt folgende Ergebnisse:

```
1:    (20) x 4             Verpackt gesamt:  80    Rest: 9
2:    (20) x 4             Verpackt gesamt:  80    Rest: 0
3:    (20) x24             Verpackt gesamt: 480    Rest: 11
4:    (20) x22             Verpackt gesamt: 440    Rest: 18
5:    (20) x 9 (30) x 8    Verpackt gesamt: 420    Rest: 15
6:    (20) x 9 (30) x 8    Verpackt gesamt: 420    Rest: 15
7:    (20) x 5             Verpackt gesamt: 100    Rest: 0
```

Nun zu den Beispielen im Einzelnen: Da sich die Beispiele 1 - 4 nur in der Länge der Füllfolge unterscheiden, sei hier nur für das erste die komplette Ausgabe abgedruckt. Auch Beispiele 5 und 6 testen den gleichen Aspekt und sind daher ebenfalls zusammengefasst.

Beispiel 1 - 4

Bei diesem Beispiel gibt es keine weiteren Besonderheiten; es gibt meist mehrere Varianten, exakt 20 zu erreichen. Daher greift an dieser Stelle noch keines der Nebenprobleme.

```
Bitte ID der Eingabe angeben:
1
[ 5,  3,  8,  3,  6,  2,  8,  4,  2,  2]
  Kippe  3 Ballons aus  2 aus =>  3 in der Schale.
[ 5,  9,  8,  3,  6,  2,  8,  4,  2,  2]
  Kippe  8 Ballons aus  3 aus => 11 in der Schale.
[ 5,  9,  1,  3,  6,  2,  8,  4,  2,  2]
  Kippe  9 Ballons aus  2 aus => 20 in der Schale.
[ 5,  3,  1,  3,  6,  2,  8,  4,  2,  2]
  Verpacke 20 Luftballons.
[ 5,  3,  1,  3,  6,  2,  8,  4,  2,  2]
  Kippe  1 Ballons aus  3 aus =>  1 in der Schale.
[ 5,  3, 11,  3,  6,  2,  8,  4,  2,  2]
  Kippe  3 Ballons aus  2 aus =>  4 in der Schale.
[ 5,  6, 11,  3,  6,  2,  8,  4,  2,  2]
  Kippe  5 Ballons aus  1 aus =>  9 in der Schale.
[ 4,  6, 11,  3,  6,  2,  8,  4,  2,  2]
  Kippe 11 Ballons aus  3 aus => 20 in der Schale.
[ 4,  6,  7,  3,  6,  2,  8,  4,  2,  2]
  Verpacke 20 Luftballons.
[ 4,  6,  7,  3,  6,  2,  8,  4,  2,  2]
  Kippe  3 Ballons aus  4 aus =>  3 in der Schale.
[ 4,  6,  7,  5,  6,  2,  8,  4,  2,  2]
  Kippe  4 Ballons aus  1 aus =>  7 in der Schale.
[ 0,  6,  7,  5,  6,  2,  8,  4,  2,  2]
  Kippe  6 Ballons aus  2 aus => 13 in der Schale.
[ 0,  0,  7,  5,  6,  2,  8,  4,  2,  2]
  Kippe  7 Ballons aus  3 aus => 20 in der Schale.
```

```
[ 0,  0,  0,  5,  6,  2,  8,  4,  2,  2]
  Verpacke 20 Luftballons.
[ 0,  0,  0,  5,  6,  2,  8,  4,  2,  2]
  Kippe  2 Ballons aus  6 aus =>  2 in der Schale.
[ 0,  0,  0,  5,  6,  0,  8,  4,  2,  2]
  Kippe  4 Ballons aus  8 aus =>  6 in der Schale.
[ 0,  0,  0,  5,  6,  0,  8,  0,  2,  2]
  Kippe  6 Ballons aus  5 aus => 12 in der Schale.
[ 0,  0,  0,  5,  0,  0,  8,  0,  2,  2]
  Kippe  8 Ballons aus  7 aus => 20 in der Schale.
[ 0,  0,  0,  5,  0,  0,  0,  0,  2,  2]
  Verpacke 20 Luftballons.
```

Beispiele 5 und 6

Diese Beispiele testen die Behandlung der Nebenprobleme 1 und 3. Hier sollte das Programm zuerst möglichst viele 20er Beutel produzieren und anschließend 30er. Die Reihenfolge der 5er- und 15er-Pakete in der Füllfolge (in 5 und 6 unterschiedlich) soll bei einer schlechten Lösung der Nebenprobleme dazu führen, dass weniger 20er Beutel als möglich produziert werden. Bei Beispiel 5 könnte ein schlechterer Algorithmus am Anfang einen Beutel mit vier mal 5 Ballons packen. Dadurch währen drei perfekte Beutel weniger möglich.

```
Bitte ID der Eingabe angeben:
5
[ 5,  5,  5,  5, 15, 15, 15, 15, 15, 15]
  Kippe  5 Ballons aus  1 aus =>  5 in der Schale.
[15,  5,  5,  5, 15, 15, 15, 15, 15, 15]
  Kippe 15 Ballons aus  1 aus => 20 in der Schale.
[15,  5,  5,  5, 15, 15, 15, 15, 15, 15]
  Verpacke 20 Luftballons.
[7 mal 5 + 15 = 20]
[15, 15, 15,  5, 15, 15, 15, 15, 15, 15]
  Kippe  5 Ballons aus  4 aus =>  5 in der Schale.
[15, 15, 15, 15, 15, 15, 15, 15, 15, 15]
  Kippe 15 Ballons aus  1 aus => 20 in der Schale.
[15, 15, 15, 15, 15, 15, 15, 15, 15, 15]
  Verpacke 20 Luftballons.
[15, 15, 15, 15, 15, 15, 15, 15, 15, 15]
  Kippe 15 Ballons aus  1 aus => 15 in der Schale.
[15, 15, 15, 15, 15, 15, 15, 15, 15, 15]
  Kippe 15 Ballons aus  1 aus => 30 in der Schale.
[15, 15, 15, 15, 15, 15, 15, 15, 15, 15]
  Verpacke 30 Luftballons.
[6 mal 15 + 15 = 30]
[ 0,  0,  0,  0,  0,  0,  0, 15, 15, 15]
  Kippe 15 Ballons aus  8 aus => 15 in der Schale.
[ 0,  0,  0,  0,  0,  0,  0,  0, 15, 15]
  Kippe 15 Ballons aus  9 aus => 30 in der Schale.
[ 0,  0,  0,  0,  0,  0,  0,  0,  0, 15]
  Verpacke 30 Luftballons.
```

Bei Beispiel 6 werden Beutel der gleichen Größe und in der gleichen Reihenfolge wie bei 5 produziert.

Beispiel 7

Dieses Beispiel testet Nebenproblem 2.

```
Bitte ID der Eingabe angeben:
7
[ 1,  1,  1,  1,  1,  1,  1,  1,  1,  1]
  Kippe  1 Ballons aus  1 aus =>  1 in der Schale.
...
[ 1,  1,  1,  1,  1,  1,  1,  1,  1,  1]
  Kippe  1 Ballons aus  1 aus => 20 in der Schale.
[ 1,  1,  1,  1,  1,  1,  1,  1,  1,  1]
  Verpacke 20 Luftballons.
...
[ 0,  0,  0,  0,  0,  0,  0,  0,  0,  0]
  Verpacke 20 Luftballons.
```

Laufzeit

Eine Laufzeitbetrachtung ist eigentlich unnötig, da nur eine LVM mit 10 Speicherfächern betrachtet wird. Bei größeren LVMs würden die Unterschiede zwischen den Lösungsansätzen zum Grundproblem jedoch deutlich werden. Daher soll an dieser Stelle eine LVM mit n Speicherfächern betrachtet werden.

Brute Force Bei n Fächern gibt es 2^n verschiedene Teilmengen, die untersucht werden müssen. Für $n = 10$ sind das nur 1024, für $n = 20$ schon 1.048.576.

Bottom-Up Hier hängt die Laufzeit von der Anzahl der Elemente in der Menge der möglichen Kombinationen aus Fächern und Ballon-Zahl ab. Die betrachteten Ballon-Zahlen lassen sich nach oben beschränken: In der hier betrachteten Implementierung werden Speicherfächer aufsummiert, bis 20 überschritten wird. Dadurch liegt die Grenze hier recht niedrig, in allen betrachteten Beispielen nie über 30. Für $n = 10$ ergeben sich $10 \cdot 30 = 300$, für $n = 20$ nur $20 \cdot 30 = 600$ betrachtete Zellen.

Man erkennt, dass der zweite Algorithmus auch für größere LVMs noch schnell arbeitet, während Brute Force deutlich länger braucht. In der Landau-Notation: Brute Force wächst mit $\mathcal{O}(2^n)$, Bottom Up mit (Pseudo-) $\mathcal{O}(n)$.

Quellcode

Der folgende Quellcode zeigt den Inhalt der Main-Methode eines C#-Programmes.
Die Berechnung der möglichen Kombinationen benötigt nur 3 Zeilen, der ganze Rest wird von der Betrachtung der Nebenfragen eingenommen.

```
//Einlesen und initialisieren der LVM
Console.WriteLine("Bitte ID der Eingabe angeben:");
string inputPath = @"luftballons#.txt".Replace("#", Console.ReadLine());
LVM lvm = new LVM(new Queue<int>(Array.ConvertAll(File.ReadAllLines(inputPath), int.Parse)));
int goal = 20;

for(int i = 0; i<10; lvm.fach(i++)) { } //Anfängliches Füllen aller Fächer
//Solange noch die Chance besteht, einen Beutel zu füllen:
while(lvm.speicherfaecher.Sum()>=goal||!lvm.speicherfaecher.Contains(0)) {
    bool[,] possible = new bool[10, lvm.speicherfaecher.Aggregate((a, b) => a>=goal ? a : a+b)+1];
    if(possible.GetLength(1)-goal<=0) { //Falls mit den aktuellen Mengen das Ziel nicht erreicht wird
      //Fach mit dem kleinsten Inhalt auskippen
        goal-=lvm.speicherfaecher.Min();
        lvm.fach(Array.IndexOf(lvm.speicherfaecher, lvm.speicherfaecher.Min()));
    } else {
        //Berechnen des Hauptproblems
        for(int i = 0; i<10; i++)
            for(int j = 0; j<possible.GetLength(1); j++)
                possible[i, j]=lvm[i]==j||i>0&&(possible[i-1, j]||j-lvm[i]>=0&&possible[i-1, j-lvm[i]]);
        //Prüfen, was die bestmögliche Anzahl ist:
        int found = Enumerable.Range(goal, possible.GetLength(1)-goal).FirstOrDefault(a => possible[9, a]);
        //Benutzte Fächer rekonstruieren
        List<int> used = new List<int>();
        for(int i = 9; i>=0; i--) {
            if(lvm[i]==found) { used.Add(i); break; }
            if(found-lvm[i]>=0&&possible[i-1, found-lvm[i]]&&(i==0||!possible[i-1, found])) { used.Add(i);
                found-=lvm[i]; }
        }
        //Fach mit dem kleinsten Inhalt auskippen
        goal-=used.Select(i => lvm.speicherfaecher[i]).Min();
        lvm.fach(Array.IndexOf(lvm.speicherfaecher, used.Select(i => lvm.speicherfaecher[i]).Min()));
    }
    if(goal<=0) { //Falls Beutel voll, Ballons verpacken
        goal=20;
        lvm.verpacken();
    }
}
```

Bewertungskriterien

- Die Laufzeit des Algorithmus ist nicht wichtig: Auch eine Brute-Force-Lösung, die systematisch alle Fächer-Kombinationen durchgeht, ist bei dieser Aufgabe in Ordnung.
- Die Mindestanforderung an eine Verpackungsstrategie: Auf keinen Fall dürfen Verpackungen mit weniger als 20 Luftballons entstehen. Akzeptabel ist aber, wenn die am Schluss übrig gebliebenen Luftballons noch verpackt werden und so eine einzige Verpackung mit weniger als 20 Luftballons entstehen kann.
- Die Nebenprobleme sollten (ggf. implizit) erkannt und begründet behandelt werden. Das ist insbesondere nicht der Fall, wenn keine besondere Strategie eingesetzt, sondern die erstbeste passende Fächerkombination genutzt wird. Die gewählten Strategien sollten zu brauchbaren Ergebnissen führen.
- Natürlich muss auch das Grundproblem gut gelöst werden. Allzu einfache Verfahren zur Bestimmung von Fächerkombinationen können leicht gute Kombinationen auslassen.
- Ein Detail: Es ist in Ordnung, wenn leere Fächer nicht weiter berücksichtigt werden, also davon ausgegangen wird, dass die Füllfolge keine 0 enthält und Fächer nur leer werden können, wenn die Füllfolge zu Ende ist. Es ist andererseits auch in Ordnung, wenn leere Fächer als Fächer mit 0 Luftballons in die Berechnungen mit einbezogen werden (das vereinfacht die Implementierung minimal), solange das nicht zu schlechteren Ergebnissen führt.
- In der Dokumentation müssen die Lösungen für zumindest einige der vorgegebenen Beispiele enthalten sein. Wenn die implementierte Strategie in manchen Fällen erkennbar nicht ideal ist, ist das nicht schlimm. In einer prerfekten Bearbeitung würden solche Nachteile angesprochen; das zeugt von einem besonders gutem Verständnis des Problems.