Aufgabe 2: Stillvolle Päckchen

Teilnahme-ID: 72940

Bearbeiter dieser Aufgabe:
Erik Donath

15. April 2024

Inhaltsverzeichnis

Lösungsidee

Als erstes werden die Kombinationen in nach ihrem Still Sortiert sodass es für jeden Still eine liste an kombinierbaren Stillen gibt. Dann geht man durch die erste liste durch und nimmt einen Still a. Mit dem Still a geht man durch die Liste für Still a und nimmt einen Still b. Mittels dieses Stilles b geht man erneut durch die Liste von Still b und nimmt einen Still c. Dann muss überprüft werden ob c mit a kompatibel ist. Still c muss also in der Liste von Still a vorhanden sein. Dann notiert man sich die Stille als 3. Kombination.

Nun werden auch die Kleidungsstücke nach ihrem Still in eigene Listen sortiert. Nun geht man für alle generierten 3. Kombinationen durch und nimmt für Still a die Liste an Kleidungstücken A des Stillen a. Das gleiche passiert auch mit den Listen B und C. Nun geht man durch liste A der Kleidungstücke und nimmt sich ein Kleidungsstück d. Nachdem man auch durch die Listen B und C gegangen ist und die Kleidungsstücke e und f ausgewählt hat, notiert man sich diese Kombination falls diese noch nicht notiert wurde. Das sind alle möglichen 3. Kleidungsstück Kombinationen. Gleichzeitig notiert man sich für jedes Kleidungsstück in einer Liste eine Referenz zu der 3. Kleidungsstück Kombination.

Dann geht man durch die Referenzen Liste der Kleidungsstücke durch und teilt man mit Rest die Anzahl an möglichen Kleiderstück Kombinationen durch die Anzahl. Der Rest wird nun in einer eigenen Liste notiert. Für jede Kombination wird nun der Teilwert für das Kleidungsstück gespeichert.

Zuletzt wird noch einmal durch die 3. Kleidungsstück Kombination Liste durchgegangen und der minimale wert der Anzahlen der Kleidungstücke für jede Kombination genommen. Diese wird dann als die allgemeine Anzahl der Kombination angesehen. Der Rest der möglicherweise entsteht wird in die Restliste notiert.

Umsetzung

Es wird eine Hashmap pmap erstellt wobei der Key der Still ist und die Value eine Liste an Stillen ist. Dann wird von 1 bis r+1 als i, dann wird bei i in der pmap die Liste um i erweitert. Anschließen wird durch die gegebenen 2. Kombinationen durch iteriert und für x in der pmap y hinzugefügt und an Stelle y x hinzugefügt.

Dann wir mit einer 3-fachen Foreach-Schleife durch alles Keys in der pmap iteriert. Der Wert wird a genannt. Dann wird durch die Liste bei a in der pmap iteriert um b zu erhalten. Zuletzt wird durch die Liste bei b in der pmap iteriert um c zu erhalten. Dann wird die 3. Kombination aus a, b und c in einer Liste gespeichert, falls diese nicht schon in der Liste vorhanden ist, oder c nicht in a vorhanden ist.

Nachdem die 3. Kombinationen gebildet wurden, wird eine Hashmap smap erstellt, wobei der Key ein Still ist und die Value eine Liste an Kleidungsstücken ist. Anschließend wird durch die gebene Liste der Kleidungsstücke iteriert und diese an Stelle des Stilles des Kleidungsstückes hinzugefügt.

Danach wird zuerst eine Hashmap cmap erstellt, wobei der Key ein Kleidungsstück ist und die Value eine Liste an Indexen ist. Als zweites wird durch alle 3. Kombinationen iteriert und dann mit einer 3-fachen Foreach-Schleife durch die smap iteriert. Die erste Schleife iteriert durch die Liste an der Stelle von a aus der Kombination in der smap. Bei der zweiten(b) und dritten(c) Schleife passiert das gleiche. Sollten nun die Stille von a, b und c unterschiedlich sein, so wird die Kombination abgespeichert. Außerdem wird für jede Kombination in der cmap der Index der Kombination gespeichert an den Stellen a, b und c.

Im Vorletzten Schritt wird nun durch die cmap iteriert. Der Wert ist ein KeyValue-Paar namens kv. Nun wird die Größe der Liste die als Value in kv gespeichert ist ermittelt. Danach wird die Anzahl des Kleidungsstückes durch die Größe ganzzahlig geteilt. Auch wird hierbei der Rest durch Modulo ermittelt. Nun wird durch die Index Liste iteriert und die Anzahl des Kleidungsstückes aktualisiert mit dem zuvor geteilten Wert. Zuletzt wird der Rest in einer Separaten Liste gespeichert.

Zuletzt wird nocheinmal durch die erstellten 3. Kleidungsstück Kombinations Listes iteriert und es wird das Minimum der Anzahlen der Kleidungsstücke ermittelt. Dieses wird dann zu der Anzahl der Kombination. Die entstehenden Reste werden in der Reste Liste gespeichert.

Beispiele

Eingabe	Ausgabe

3 2
1 2
 	
1 1 3
1 2 2
2 2 3
3 2 3
	Sorten Anzahl: 3
	Still Anzahl: 2
	
	Kombinationen:
	  1 <> 2
	
	Dinge:
	  1 + 1 => 3
	  1 + 2 => 2
	  2 + 2 => 3
	  3 + 2 => 3
	Funktion took 9100 nanoseconds (0 miliseconds).
	
	Packete:
	  1 * {(1 1) + (2 2) + (2 3)}
	  1 * {(2 1) + (2 2) + (2 3)}
	
	Rest:
	  1 * (2 2)

	3 * (3 2) 2 * (1 1) 1 * (1 2)
3 4 1 2 2 3 3 4 1 1 48 1 2 55 1 3 76 1 4 27 2 1 39 2 2 46 2 3 29 2 4 37 3 1 57 3 2 38 3 3 28 3 4 19	Sorten Anzahl: 3 Still Anzahl: 4 Kombinationen: 1 <> 2 2 <> 3 3 <> 4 Dinge: 1 + 1 => 48 1 + 2 => 55 1 + 3 => 76 1 + 4 => 27 2 + 1 => 39 2 + 2 => 46 2 + 3 => 29 2 + 4 => 37 3 + 1 => 57 3 + 2 => 38 3 + 3 => 28 3 + 4 => 19 Funktion took 24200 nanoseconds (0 miliseconds). Packete: 9 * {(1 1) + (1 2) + (1 3)} 5 * {(1 1) + (1 2) + (2 3)} 6 * {(1 1) + (1 3) + (2 2)} 7 * {(1 2) + (1 3) + (2 1)} 5 * {(1 1) + (2 2) + (2 3)} 5 * {(1 2) + (2 1) + (2 3)} 6 * {(1 3) + (2 1) + (2 2)} 5 * {(2 1) + (2 2) + (2 3)} 4 * {(2 1) + (2 2) + (3 3)} 4 * {(2 1) + (2 3) + (3 2)} 5 * {(2 2) + (2 3) + (3 1)} 4 * {(2 1) + (3 2) + (3 3)} 4 * {(2 2) + (3 1) + (3 3)} 4 * {(2 3) + (3 1) + (3 2)} 4 * {(3 1) + (3 2) + (3 3)} 4 * {(3 1) + (3 2) + (4 3)} 4 * {(3 1) + (3 3) + (4 2)} 4 * {(3 2) + (3 3) + (4 1)} 4 * {(3 1) + (4 2) + (4 3)} 4 * {(3 2) + (4 1) + (4 3)} 4 * {(3 3) + (4 1) + (4 2)} 4 * {(4 1) + (4 2) + (4 3)} Rest: 38 * (3 1) 13 * (2 1) 27 * (1 2) 25 * (3 2) 9 * (2 3) 29 * (2 4) 23 * (2 2) 15 * (1 4) 52 * (1 3) 19 * (3 4)

	23 * (1 1) 16 * (3 3)
3 9 1 2 1 3 1 4 1 5 1 6 1 8 1 9 2 3 2 5 2 6 2 8 2 9 3 5 3 7 3 9 4 5 6 8 7 9 8 9 1 1 1 1 2 1 1 3 1 1 6 1 1 7 2 2 4 2 2 5 2 2 2 1 2 6 3 2 7 4 2 8 1 3 5 3 3 2 3 3 8 2 3 9 4 3 7 1	Sorten Anzahl: 3 Still Anzahl: 9 Kombinationen: 1 <> 2 1 <> 3 1 <> 4 1 <> 5 1 <> 6 1 <> 8 1 <> 9 2 <> 3 2 <> 5 2 <> 6 2 <> 8 2 <> 9 3 <> 5 3 <> 7 3 <> 9 4 <> 5 6 <> 8 7 <> 9 8 <> 9 Dinge: 1 + 1 => 1 1 + 2 => 1 1 + 3 => 1 1 + 6 => 1 1 + 7 => 2 2 + 4 => 2 2 + 5 => 2 2 + 2 => 1 2 + 6 => 3 2 + 7 => 4 2 + 8 => 1 3 + 5 => 3 3 + 2 => 3 3 + 8 => 2 3 + 9 => 4 3 + 7 => 1 Funktion took 50800 nanoseconds (0 miliseconds). Packete: 0 * {(1 1) + (2 2) + (2 3)} 0 * {(1 1) + (2 2) + (5 3)} 0 * {(1 1) + (2 3) + (5 2)} 0 * {(1 1) + (2 3) + (6 2)} 0 * {(1 1) + (2 2) + (8 3)} 0 * {(1 1) + (2 3) + (8 2)} 0 * {(1 1) + (2 2) + (9 3)} 0 * {(1 1) + (4 2) + (5 3)} 0 * {(1 1) + (5 2) + (5 3)} 0 * {(1 1) + (6 2) + (8 3)} 0 * {(1 1) + (8 2) + (8 3)} 0 * {(1 1) + (8 2) + (9 3)} 0 * {(2 1) + (2 2) + (2 3)} 0 * {(2 2) + (2 3) + (3 1)} 0 * {(2 1) + (2 2) + (5 3)}

```
  0 * {(2 1) + (2 3) + (5 2)}
  0 * {(2 1) + (2 3) + (6 2)}
  0 * {(2 2) + (2 3) + (6 1)}
  0 * {(2 1) + (2 2) + (8 3)}
  0 * {(2 1) + (2 3) + (8 2)}
  0 * {(2 1) + (2 2) + (9 3)}
  0 * {(2 2) + (3 1) + (5 3)}
  0 * {(2 3) + (3 1) + (5 2)}
  0 * {(2 2) + (3 1) + (9 3)}
  0 * {(2 1) + (5 2) + (5 3)}
  0 * {(2 3) + (6 1) + (6 2)}
  0 * {(2 1) + (6 2) + (8 3)}
  0 * {(2 2) + (6 1) + (8 3)}
  0 * {(2 3) + (6 1) + (8 2)}
  0 * {(2 1) + (8 2) + (8 3)}
  0 * {(2 1) + (8 2) + (9 3)}
  0 * {(3 1) + (5 2) + (5 3)}
  0 * {(3 1) + (7 2) + (7 3)}
  0 * {(3 1) + (7 2) + (9 3)}
  0 * {(6 1) + (6 2) + (8 3)}
  0 * {(6 1) + (8 2) + (8 3)}
  0 * {(7 1) + (7 2) + (7 3)}
  0 * {(7 1) + (7 2) + (9 3)}

Rest:
  1 * (3 7)
  3 * (3 2)
  1 * (1 1)
  1 * (2 2)
  3 * (3 5)
  2 * (2 5)
  2 * (3 8)
  3 * (2 6)
  4 * (3 9)
  1 * (2 8)
  1 * (1 3)
  1 * (1 2)
  1 * (1 6)
  2 * (2 4)
  4 * (2 7)
  2 * (1 7)
```

Quellcode

Solution.h

```cpp
#pragma once
#include <vector>

typedef int32_t     i32;
typedef uint32_t    u32;
typedef const char  cstr;

struct Pair {
    u32 x, y; // x; Still 1, y: Still 2
```

```
};

struct Combination {
    u32 a, b, c;

    inline const bool operator==(const Combination& other) const {
        return (
            (a == other.a && b == other.b && c == other.c) ||
            (a == other.a && b == other.c && c == other.b) ||

            (a == other.b && b == other.c && c == other.a) ||
            (a == other.b && b == other.a && c == other.c) ||

            (a == other.c && b == other.a && c == other.b) ||
            (a == other.c && b == other.b && c == other.a)
        );
    }
};

struct Thing {
    u32 i, j, n; // i: Sorte, j: Still, n: Anzahl

    inline bool operator==(const Thing& other) {
        return (
            this->i == other.i &&
            this->j == other.j
        );
    }
    inline bool operator==(Thing& other) {
        return (
            this->i == other.i &&
            this->j == other.j
            );
    }
};

struct Packet {
    Thing a, b, c; // Things werden anzahl = 1 haben müssen.
    u32 n;

    Packet(Thing& a, Thing& b, Thing& c) : a(a), b(b), c(c), n(1) { }
    Packet(Combination& combi) {
        a.i = combi.a;
        b.i = combi.b;
        c.i = combi.c;
        n = 1;
    }
    inline void SetThingCount(Thing& t, u32 n) {
        if (a.i == t.i && a.j == t.j) a.n = n;
        if (b.i == t.i && b.j == t.j) b.n = n;
        if (c.i == t.i && c.j == t.j) c.n = n;
    }

    inline const bool hasThing(const Thing& thing) const {
        return (
            (this->a.i == thing.i && this->a.j == thing.j) ||
            (this->b.i == thing.i && this->b.j == thing.j) ||
            (this->c.i == thing.i && this->c.j == thing.j)
            );
    }
    inline bool operator==(const Packet& other) {
```

```
        return (
            (this->a == other.a && this->b == other.b && this->c == other.c) ||
            (this->a == other.a && this->b == other.c && this->c == other.b) ||

            (this->b == other.a && this->c == other.b && this->a == other.c) ||
            (this->b == other.a && this->c == other.c && this->a == other.b) ||

            (this->c == other.a && this->a == other.b && this->b == other.c) ||
            (this->c == other.a && this->a == other.c && this->b == other.b)
        );
    }
};

struct Solution {
    std::vector<Packet>* packets;
    std::vector<Thing>* rest;

    Solution(std::vector<Packet>* packets, std::vector<Thing>* rest) : packets(pack-
ets), rest(rest) { }

    ~Solution() {
        delete packets, rest;
    }
};

const Solution Solve(const std::vector<Pair>& pairs, const std::vector<Thing>& things,
u32 s, u32 r);
```

Solution.cpp

```
#include <unordered_map>
#include <vector>

#include "Solution.h"

#include <iostream>

template<class Container, class Object>
inline bool contains(Container& container, const Object& value) {
      return !container.empty() && std::find(container.begin(), container.end(),
value) != container.end();
}
template<class Container, class Object>
inline u32 getIndex(Container& container, const Object& value) { // If not in con-
tainer it will be the lenght of the container. [2, 3] search for 1 => 3
      return std::distance(container.begin(), std::find(container.begin(), con-
tainer.end(), value));
}
template<class Container, class Object>
inline Object* getElement(Container& container, const Object& value) {
      u32 index = getIndex(container, value);
      if (index == container.end() - container.begin()) return nullptr;
      return &container[index];
}

static inline u32 minimum(u32 a, u32 b) {
      return (a < b) ? a : b;
}
```

```
static inline u32 minimum(u32 a, u32 b, u32 c) {
	return minimum(minimum(a, c), minimum(b, c));
}
static void addThing(std::vector<Thing>& rest, Thing& thing, u32 n) {
	Thing* t = getElement(rest, thing);
	if(t == nullptr) {
		Thing t(thing);
		t.n = n;
		rest.push_back(t);
	}
	else {
		t->n += n;
	}
}

static const std::vector<Combination>* generateCombinations(const std::vector<Pair>&
pairs, u32 r) {
	std::unordered_map<u32, std::vector<u32>> pmap;

	for (int i = 1; i <= r; i++) pmap[i].push_back(i);
	for (const Pair& pair : pairs) {
		pmap[pair.x].push_back(pair.y);
		pmap[pair.y].push_back(pair.x);
	}

	std::vector<Combination>* combinations = new std::vector<Combination>;
	for (auto& kv : pmap) {
		u32 i = kv.first;
		std::vector<u32>& a = kv.second;
		for (u32 j : a) {
			std::vector<u32>& b = pmap[j];
			for (u32 k : b) {
				Combination c = Combination(i, j, k);
				if (contains(a, k) && !contains(*combinations, c)) {
					combinations->push_back(c);
				}
			}
		}
	}
	return combinations;
}


const Solution Solve(const std::vector<Pair>& pairs, const std::vector<Thing>& things,
u32 s, u32 r) {
	std::vector<Packet>* packets = new std::vector<Packet>;
	std::vector<Thing>* rest = new std::vector<Thing>;

	std::unordered_map<u32, std::vector<Thing>> smap;
	std::unordered_map<Thing*, std::vector<u32>> cmap;

	for (const Thing& thing : things)
		smap[thing.j].push_back(thing);

	const std::vector<Combination>* combinations = generateCombinations(pairs, r);
	for (const Combination& c : *combinations) {
		for (Thing& a : smap[c.a]) {
			for ( Thing& b : smap[c.b]) {
				for (Thing& c : smap[c.c]) {
					if (a == b || a == c || b == c) continue;
					if (a.i == b.i || a.i == c.i || b.i == c.i) continue;
```

```
                                    Packet p = Packet(a, b, c);
                                    if (contains(*packets, p)) continue;

                                    u32 index = packets->size();
                                    packets->push_back(p);

                                    if (!contains(cmap[&a], index)) cmap[&a].push_back(in-
dex);
                                    if (!contains(cmap[&b], index)) cmap[&b].push_back(in-
dex);
                                    if (!contains(cmap[&c], index)) cmap[&c].push_back(in-
dex);
                              }
                        }
                  }
            }

            for (auto& kv : cmap) {
                  Thing t = *kv.first;
                  std::vector<u32> packetIndexes = kv.second;

                  Thing rt = Thing(t);
                  u32 n = t.n / packetIndexes.size();
                  rt.n = t.n % packetIndexes.size();

                  for (u32 index : packetIndexes) {
                        Packet& p = (*packets)[index];
                        p.SetThingCount(t, n);
                  }
                  if(rt.n > 0)
                        rest->push_back(rt);
            }

            for (Packet& p : *packets) {
                  u32 m = minimum(p.a.n, p.b.n, p.c.n);
                  u32 ar = p.a.n - m, br = p.b.n - m, cr = p.c.n;
                  p.n = p.a.n = p.b.n = p.c.n = m;

                  if (ar > 0) addThing(*rest, p.a, ar);
                  if (br > 0) addThing(*rest, p.b, br);
                  if (cr > 0) addThing(*rest, p.c, cr);
            }

            delete combinations;
            return Solution(packets, rest);
}
```