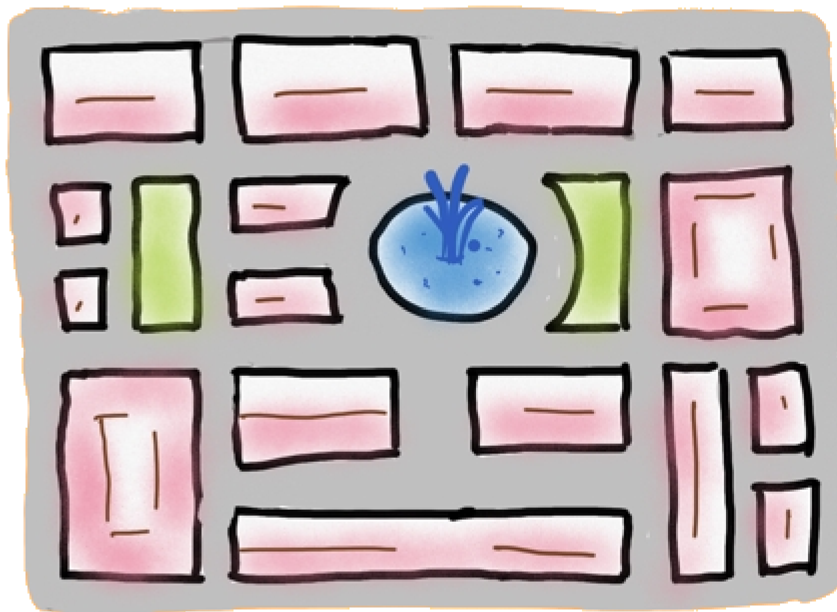
Problemstellung

In einem heißen Sommer möchte ein Eisproduzent in einer Stadt eine perfekte Versorgung der



Stadtplan (eigenes Werk)

Bevölkerung mit Eis sicherstellen (natürlich, um möglichst viel zu verdienen) und stellt dazu Eisstände auf. Kein Bürger und keine Bürgerin der Stadt soll zu weit laufen müssen. Daher muss an jeder Straßenecke entweder ein Eisstand stehen oder an mindestens einer der benachbarten Straßenkreuzungen ein Eisstand zu finden sein.

Aufgabe:

1. *Finde für die oben abgebildete Stadt geeignete Standorte für die Eisstände, so dass möglichst wenige Eisstände benötigt werden, da das der Firma Kosten spart.*
   Anmerkung: Auch die Straße um alle Häuser außen herum zählt als Straße und damit die äußeren Ecken als Kreuzungen.

Modellierung

Die Ausgangssituation soll nun als Graph modelliert werden.

Modellierung

Knoten:

Kanten:

Dominierende Menge

CC BY NC SA

Kopiervorlage für die Eisstände

Quelle: https://pixabay.com/de/vectors/eis-waffel-dessert-sommer-s%C3%BC%C3%9F-1432278/ (Pixabay-Lizenz - keine Bildnachweis notwendig)

Weiterführende Fragen

Ihr habt sicher eine Lösung gefunden, die die Abdeckung der Stadt mit Eisständen sicherstellt. Aber ist das die beste Lösung? Vielleicht geht es ja mit weniger Eisständen. Es ist kein cleverer Algorithmus bekannt, der gezielt die richtigen Knoten auswählt. Man müsste daher alle möglichen Teilmengen der Knoten daraufhin testen, ob sie eine überdeckende Menge darstellen, und unter den überdeckenden Mengen die mit der geringsten Anzahl an Knoten auswählen.[1]

Aufgaben

2. *Berechne, wie viele mögliche Teilmengen der Knoten es beim Einstiegsbeispiel gibt. Beachte, dass jeder Knoten zur Teilmenge dazugehören kann oder auch nicht. Wie ändert sich die Anzahl der Möglichkeiten, wenn ein Knoten hinzukommt?*
3. *Führe im Simulationsmodus des Programms Graphentester den Algorithmus "Dominierende Menge (Vollständig)" bei den Graphen "graph_knotenX.csv" (X=5-10) aus und bestimme den Zeitbedarf. Stelle dabei die Geschwindigkeit so ein, dass beim Graphen mit 5 Knoten ca. 3 Sek. benötigt werden.*
   Untersuche, wie sich der Zeitbedarf durch einen zusätzlichen Knoten verändert.
4. *Stelle auf maximale Geschwindigkeit um und führe den Algorithmus beim Graph mit 10 Knoten erneut aus. Sage den Zeitbedarf für den Graphen mit 15 Knoten vorher. Überprüfe deine Vorhersage. Berechne, wie lange es für das Einstiegsbeispiel dauern würde.*

Näherungslösung

Wenn es zu lange dauert, die beste Lösung zu finden, ist man manchmal auch mit einer guten Lösung zufrieden: Ein oder zwei Eisstände mehr als bei der optimalen Lösung sind in einer Stadt kein Problem.

5. *Beschreibe, wie du vorgegangen bist, um deine Lösung im Einstiegsbeispiel zu finden.*

Die Greedy-Strategie (gieriger Algorithmus) ist ein Ansatz, um Näherungslösungen zu bestimmen. Sie zeichnen sich dadurch aus, dass sie schrittweise die Möglichkeit auswählen, die zum Zeitpunkt der Wahl den größten Gewinn bzw. das beste Ergebnis verspricht.
Beim Problem der dominierenden Menge fügt man schrittweise der gesuchten Teilmenge Knoten hinzu, bis sie eine überdeckende Teilmenge darstellt. Hat man einen Knoten einmal hinzugefügt, nimmt man diese Entscheidung nicht mehr zurück, auch wenn die Entscheidung nicht optimal war.

6. *Analysiere, welcher Knoten als nächstes hinzugefügt werden sollte. Notiere zunächst deine Überlegungen und kontrolliere die Ergebnisse dann mit dem Graphentester "Dominierende Menge (Greedy (a-i))" anhand der Graphen "graph_domknotenXX". Dort sind die dominierenden Knoten der optimalen Lösung mit einem Stern (*) markiert. Du kannst also untersuchen, wie gut eine Strategie funktioniert:*
   a) der Knoten mit den meisten Nachbarknoten.
   b) der Knoten mit den wenigsten Nachbarknoten.
   c) der Knoten, der die meisten Knoten neu überdeckt.
   d) der Knoten der die wenigsten Knoten neu überdeckt.
   e) ein Knoten, der von einem schon ausgewählten Knoten die Entfernung 3 hat.
   f) ein noch nicht überdeckter Kn. mit Entfernung 2 von einem schon ausgewählten Kn.
   g) ein noch nicht überdeckter Kn. mit Entfernung 3 von einem schon ausgewählten Kn.
   h) der noch nicht überdeckte Knoten, der von möglichst vielen schon ausgewählten Knoten die Entfernung 3 hat.
   i) der noch nicht überdeckte Knoten mit der größten Entfernung von allen schon ausgewählten Knoten.
7. *Wende die erfolgversprechenden Strategien auf das Einstiegsbeispiel an.*
8. *Begründe, warum die Näherungslösungen viel schneller als die optimale Lösung gefunden werden.*

1 Eine kleine Verbesserung erhält man, wenn man während des Testens weiterer Teilmenge, die Anzahl der Knoten der bisher besten Lösung beachtet. Eine grundlegende Verbesserung bringt das aber nicht.

CC BY NC SA

Pseudocode des Greedy-Algorithmus zur Bestimmung der dominierenden Menge

Sei K die Menge alle Knoten und M die Menge der ausgewählten Knoten.

```
Solange es noch nicht überdeckte Knoten gibt, wiederhole
  Bestimme den besten neu hinzuzufügenden Knoten (je nach Strategie)
  Füge den besten Knoten M hinzu
  Kennzeichne alle seine Nachbarknoten als überdeckt
Ende Wiederhole

Eine mögliche Strategie für "Bestimme den Besten"
  Wiederhole für jeden Knoten, der nicht in M liegt
    Bestimme die Anzahl der noch nicht überdeckten Nachbarknoten
    Falls der Knoten selbst noch nicht überdeckt ist,
      addiere 1 zur Anzahl
  Ende Wiederhole
  Nimm den Knoten mit der höchsten Anzahl
```

Quelltext des Greedy-Algorithmus zur Bestimmung der dominierenden Menge

```
if (g.getAnzahlKnoten()==0) {
  return;
}
List<Knoten> knoten;
List<Knoten> nachbarn;

knoten = g.getAlleKnoten();

while(knoten.size() > 0) {
  Knoten bester = bestimmeBesten();
  bester.setMarkiert(true);
  bester.setBesucht(false);
  nachbarn = g.getNachbarknoten(bester,
                     kn->!kn.isMarkiert() && !kn.isBesucht());
  for(Knoten k : nachbarn) {
    k.setBesucht(true);
  }
  knoten = g.getAlleKnoten(kn->!kn.isMarkiert() && !kn.isBesucht());
}// end of while


// Bestimme Besten z.B. mit:
private Knoten bestimmeBesten() {
  List<Knoten> knoten = g.getAlleKnoten(k->!k.isMarkiert());
  for(Knoten k : knoten) {
    List<Knoten> nachbarn = g.getNachbarknoten(k,
                       kn -> !kn.isMarkiert() && !kn.isBesucht());
    k.setWert(nachbarn.size());
    if(!k.isMarkiert() && !k.isBesucht()) {
      k.setWert(k.getIntWert()+1);
    }
  }
  knoten.sort(Comparator.comparing(Knoten::getIntWert).reversed());
  Knoten bester = knoten.get(0);
  return bester;
}
```

CC BY NC SA