Algorithmen und Datenstrukturen 1

Ausgearbeitetes Übungsblatt 6

Datum: 16. Juni 2005
Erstellt mit LaTeX

Aufgabe 6.1

Aufgabenstellung:

Wir betrachten das Rucksack-Problem, das sich für den Weihnachtsmann mit vier verschiedenen Geschenken stellt. Die folgende Tabelle gibt jeweils für jedes der Geschenke G_1, G_2, G_3 und G_4 ihr Gewicht und ihren Wert an. Der Rucksack des Weihnachtsmanns kann ein Gewicht von 100 Einheiten transportieren.

Geschenk	G_1	G_2	G_3	G_4
Gewicht g_i	50	30	25	15
Wert w_i	10	8	6	1

Sei $F = \langle F_1, \ldots, F_4 \rangle$ die Folge der vier Gegenstände, absteigend sortiert nach dem Wert des Quotienten $\frac{w_i}{g_i}$. In einem *Branch-and-Bound* Algorithmus wird das Problem in Teilprobleme zerlegt, indem man in der durch F festgelegten Reihenfolge die Werte der Entscheidungsvariablen x_i der Gegenstände fixiert.
Dadurch ergibt sich ein Branch-and-Bound Baum. Für jeden Knoten v des Baums werden eine obere und eine untere Schranke für die im Teilbaum mit Wurzel v enthaltene beste Lösung bestimmt. Seien nun die Variablen x_1 bis x_k für $1 \leq k \leq 4$ schon fixiert, wobei x_i für $1 \leq k \leq 4$ den Wert 1 hat, falls Geschenk F_i eingepackt wird, und 0 sonst.
Die untere Schranke erhält man, indem man alle Gegenstäde, deren Variable noch nicht festgelegt ist, in der durch F gegebenen Reihenfolge durchläuft und den aktuellen Gegenstand einpackt, falls noch Platz im Rucksack ist. Man erhält eine gültige Lösung für das Problem, deren Wert die untere Schranke ist.
Man erhält die obere Schranke, indem man ebenfalls alle Gegenstände, deren Variable noch nicht festgelegt ist, in der durch F gegebenen Reihenfolge durchläuft. Man packt aber nun alle Gegenstände ein, bis man zu dem ersten Gegenstand F_i kommt, der nicht mehr in den Rucksack passt. Sei r die noch freie Kapazität des Rucksacks. Dann zählt man $r \cdot \frac{w_i}{g_i}$ noch zu dem Wert der Gegenstände im Rucksack dazu.
Zeichnen Sie den Branch-and-Bound Baum für das Problem und schreiben Sie an jeden Knoten den Wert der unteren und oberen Schranke. Nehmen Sie an, dass beim Fixieren einer neuen Variable immer zuerst das Kind betrachtet wird, bei dem die Variable auf 1 gesetzt wird und dass Sie den Baum mittels Tiefensuche durchmustern. Achten Sie darauf, dass Sie nur jene Teile des Baums zeichnen, die zur Berechnung der optimalen Lösung nötig sind.

Lösung:

Zuerst wird der Nutzen der vier Geschenke berechnet:

Geschenk	G_1	G_2	G_3	G_4
Gewicht g_i	50	30	25	15
Wert w_i	10	8	6	1
Nutzen $\frac{w_i}{g_i}$	1/5	4/15	6/25	1/15

Daher ergibt sich: $F = \langle G_2, G_3, G_1, G_4 \rangle$. Nun wird eine untere und eine obere Schranke berechnet:

- Untere Schranke: eingepackt werden G_2, G_3 sowie G_4; ergibt als Gewicht 70 und als Wert (und somit als Schranke) 15. Die neue globale untere Schranke ist damit ebenfalls 15 (ist zuvor nicht gesetzt worden).
- Obere Schranke: eingepackt werden G_2 und G_3, G_1 passt nicht mehr; der Wert ist nun 14, das Gewicht 55, der Rucksack hat daher noch eine freie Kapazität von $r = 45$. Daher ist die obere Schranke $14 + r \cdot \frac{w_i}{g_i} = 23$.

Nun wird G_2 fixiert, im ersten Fall nicht aufgenommen:

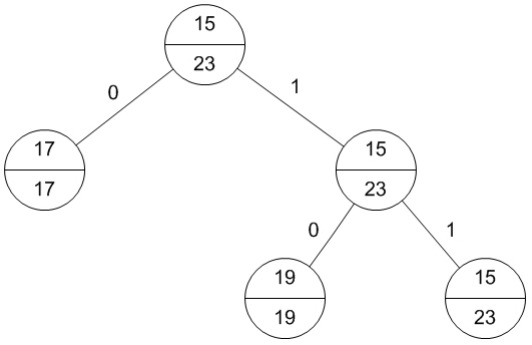
- Für die untere Schranke werden G_1, G_3 und G_4 aufgenommen; Gewicht ist dann 90, Wert (untere Schranke) ist 17, wird daher auch neue globale untere Schranke.
- Für die obere Schranke werden dieselben Gegenstände aufgenommen, ergibt auch den Wert 17. Untere Schranke ist gleich oberer Schranke, also kann dieser Ast des Branch-and-Bound-Baums abgehakt werden.

Nun wird G_2 aufgenommen (zweiter Ast); es ergibt sich die Elternlösung, also gleiche untere bzw. obere Schranke. Nun wird G_3 fixiert, zunächst nicht aufgenommen:

- Aufnahme von G_2, G_1 und G_4 liefert Gewicht 95, Wert (untere Schranke) 19, daher neue globale untere Schranke 19.
- Obere Schranke ist auch 19 (dieselben Gegenstände werden aufgenommen); untere Schranke ist gleich oberer Schranke, daher ist auch dieser Ast nicht mehr zu verbessern.

Wird G_3 dagegen aufgenommen, ergibt sich die Elternlösung; hier kann keine Lösung mehr gefunden werden, welche besser sein könnte als die gerade ermittelte Lösung, also ist auch dieser Ast des Baums abgeschlossen. Insgesamt ergibt sich also als beste Lösung das Einpacken von G_1, G_2 und G_4 mit Gesamtgewicht 95 und Wert 19.

Der entsprechende Branch-and-Bound-Baum sieht wie folgt aus:



Die obere Zahl in jedem Knoten gibt jeweils die untere Schranke an, die untere Zahl die obere Schranke.

Aufgabe 6.2

Aufgabenstellung:

Ein Betrieb stellt handgefertigte Gartentore aus Metall her. Die Bestandteile für jedes Gartentor werden jeweils von einem einzelnen Schmied zugeschnitten, gebogen, geschweißt und nachbearbeitet. Aus organisatorischen Gründen kann die Herstellung eines einzelnen Gartentores nicht auf mehrere Schmiede verteilt werden.
Je nach Komplexität eines Gartentores ist eine unterschiedliche Anzahl von Arbeitsstunden nötig, um die Fertigung zu vollenden. Momentan liegen zehn Bestellungen vor, die alle in genau einer Woche ausgeliefert werden müssen. Das bedeutet, dass noch genau 40 Arbeitsstunden Zeit sind.
Die nachstehende Tabelle zeigt die zur Herstellung der zehn bestellten Gartentore jeweils benötigte Anzahl von Arbeitsstunden.

Gartentor	K_1	K_2	K_3	K_4	K_5	K_6	K_7	K_8	K_9	K_{10}
Arbeitsstunden	38	23	6	2	22	25	32	36	39	40

Verwenden Sie die First-Fit-Heuristik aus der Vorlesung, um die Gartentore auf möglichst wenige Schmiede zu verteilen (unter Berücksichtigung der verbleibenden Zeit), sodass trotzdem alle Gartentore innerhalb der 40 Arbeitsstunden bis zum geforderten Auslieferungstermin fertig werden.
Geben Sie für jeden Schmied an, welche Gartentore er anfertigt und wie viele Stunden er in dieser Woche insgesamt arbeitet.

Lösung:

Schmied	**Gartentore**	**Arbeitszeit**
1	K_1, K_4	40
2	K_2, K_3	29
3	K_5	22
4	K_6	25
5	K_7	32
6	K_8	36
7	K_9	39
8	K_{10}	40

Aufgabe 6.3

Aufgabenstellung:

Für die fünf Beispiele auf dem Übungsblatt zu einer besonders anspruchsvollen Speziallehrveranstaltung aus „Algorithmen und Datenstrukturen“ erhalten Sie jeweils eine unterschiedliche Anzahl von Bonuspunkten, aber die Beispiele verursachen natürlich auch einen unterschiedlich hohen Zeitaufwand. In der folgenden Tabelle sind diese Daten angegeben:

Beispiel	**Bonuspunkte**	**Zeit in Minuten**
Algorithmus A	7	40
Algorithmus B	3	10
Algorithmus C	10	60
Algorithmus D	4	20
Algorithmus E	13	90

Leider sind Sie wieder einmal spät dran und haben nur noch drei Stunden Zeit, bis die Übung beginnt. Geben Sie einen Algorithmus im Pseudocode an, der schnell berechnet, welche Beispiele Sie in der verfügbaren Zeit lösen sollten, um eine möglichst hohe Punktezahl zu erzielen.
Bedenken Sie dabei, dass Sie es schon eilig haben! Eine Lösung mittels Enumeration ist also wegen des exponentiellen Zeitaufwands nicht geeignet. Veranschaulichen Sie die Schritte zur Lösung anhand der gegebenen Beispielinstanz.

Lösung:

Im folgenden Algorithmus werden die folgenden Variablen verwendet:

- Sei $t[i]$, i aus $\{A, B, C, D, E\}$, die Zeit, welche für einen Algorithmus benötigt wird.
- Sei $p[i]$, i aus $\{A, B, C, D, E\}$, die Anzahl der Punkte, welche für diesen Algorithmus erreichbar sind.
- Sei *maxzeit* die Zeit, welche höchstens zur Verfügung steht.
- Sei $n[i]$, i aus $\{A, B, C, D, E\}$, der Nutzen eines Algorithmus als Quotient $\frac{p[i]}{t[i]}$.

```
WähleAlgorithmen {
  berechne für alle Algorithmen n[i] = p[i] / t[i];
  speichere die Algorithmen absteigend sortiert nach n[i] in F;
  L = {};
  gesamtdauer = 0;
  gesamtpunkte = 0;
  für alle Algorithmen X aus F {
    dauer = gesamtdauer + t[X];
    wenn dauer <= maxzeit {
      gesamtdauer = dauer;
      L = L + X;
      gesamtpunkte = gesamtpunkte + p[X];
    }
  }
}
```

Die Mengenoperation $L + X$ steht dabei vereinfachend für $L \cup X$.

Der vorliegende Algorithmus probiert nach Greedy-Schema für alle Aufgaben, ob diese noch in das zur Verfügung stehende Zeitfenster passt oder nicht. Wenn ja, dann wird der Algorithmus ausgewählt. Ergebnis ist die Menge L der Algorithmen, welche nach diesem Verfahren die beste gefundene Lösung darstellt, um in der zur Verfügung stehenden Zeit die meisten Punkte zu ergattern.

Die Ausführung sieht dabei wie folgt aus:

- Für die Algorithmen wird der Nutzen berechnet:
  - Algorithmus A: $7/40 = 0,715$
  - Algorithmus B: $3/10 = 0,3$
  - Algorithmus C: $1/6 \approx 0,16$
  - Algorithmus D: $1/5 = 0,2$
  - Algorithmus E: $13/90 \approx 0,14$

- B wird ausgewählt; Gesamtdauer: 10 min, erreichbare Punkte: 3
- D wird ausgewählt; Gesamtdauer: 30 min, erreichbare Punkte: 7
- A wird ausgewählt; Gesamtdauer: 70 min, erreichbare Punkte: 14
- C wird ausgewählt; Gesamtdauer: 130 min, erreichbare Punkte: 24

Die gefundene Lösung ist also $\{A, B, C, D\}$ mit einer Gesamtdauer von 130 Minuten bei 24 erreichbaren Punkten. Da E allerdings nicht betrachtet wurde, lässt sich dieses Ergebnis noch gezielt verbessern, indem man probiert, ob man einen der ausgewählten Algorithmen durch E ersetzen kann. Das kann beispielsweise durch folgenden Algorithmus erfolgen:

```
VerbessereAuswahl {
  bestdauer = gesamtdauer;
  best = L;
  bestpunkte = gesamtpunkte;
  für alle X aus {A,B,C,D} {
    dauer = gesamtdauer - t[X] + t[E];
    punkte = gesamtpunkte - p[X] + p[E];
    wenn punkte>bestpunkte && dauer<=maxzeit {
      best = (L \ X) + E;
      bestdauer = dauer;
      bestpunkte = punkte;
    }
  }
}
```

$L \backslash X$ steht für die Mengendifferenz („L ohne X“).

Mit dieser Verbesserung wird schließlich Algorithmus A durch Algorithmus E ersetzt, sodass man die Lösung $\{B, C, D, E\}$ mit einer Dauer von 180 Minuten bei einer Höchstpunktezahl von 30 erhält.

Aufgabe 6.4

Aufgabenstellung:

Geben Sie einen Branch-and-Bound-Algorithmus für das Problem aus Aufgabe 6.3 in Pseudocode an. Wenden Sie dann diesen Algorithmus auf die angegebene Beispielinstanz an und veranschaulichen Sie dabei alle Schritte auf dem Weg zur Lösung.

Lösung:

Im Folgenden seien die Algorithmen der Einfachheit halber mit 1, 2, 3, 4, 5 bezeichnet; der vorliegende Algorithmus funktioniert analog zu Aufgabe 6.1 (es handelt sich wieder um ein Rucksackproblem); seien des weiteren $t[i]$ die Zeit, die für den Algorithmus i benötigt wird, und $p[i]$ die Punkte, welche für diesen Algorithmus erreichbar sind:

```
WähleAlgorithmen {
  maxzeit = 180;

  // Nutzen berechnen
  für i = 1..5 {
    n[i] = p[i] / n[i];
  }

  Sortiere A nach n[i] absteigend

  // alle Algorithmen als "nicht fixiert" markieren
  für i = 1..4 {
    P[i] = -1;
  }

  // Anzahl der bereits fixierten Algorithmen
  P[0] = 0;

  gesamtpunkte = 0;  // Punkteanzahl der bisher besten Lösung
  gesamtdauer = 0;   // Dauer der bisher besten Lösung

  schranke = 0;   // globale untere Schranke
```

```
  X = {P};  // X ist die Menge der zu bearbeitenden Probleme

  solange x != {} {
    P = erstes Element von X;
    P‘ = P;
    X = X \ P;

    // Bisher eingepackte Gegenstände erfassen
    dauer1 = 0;
    punkte1 = 0;
    für i = 1..P[0] - 1 {
      wenn P[i] == 1 {
        dauer1 += t[i];
        punkte1 += p[i];
      }
    }
    dauer2 = dauer1;
    punkte2 = punkte1;

    // untere schranke berechnen
    // (alles wird eingepackt, wenn's reinpasst)
    für i = P[0]..5 {
      wenn dauer1 + t[i] < maxzeit {
        dauer1 += t[i];
        punkte1 += p[i];
        P‘[i] = 1;
      }
      sonst {
        P‘[i] = 0;
      }
    }
    // untere Schranke: punkte1

    // obere schranke berechnen
    // (es wird so lange eingepackt, bis ein Gegenstand nicht reinpasst)
    für i = P[0]..5 {
      wenn dauer1 + t[i] < maxzeit {
        dauer2 += t[i];
        punkte2 += p[i];
      }
      sonst {
        r = maxzeit - dauer2;
        punkte2 += r * p[i] / t[i];
        break;
      }
    }
    // obere schranke: punkte2

    wenn schranke < punkte1 {
      // beste lösung gefunden
      schranke = punkte1;
      U = P;

      wenn punkte1 != punkte2 {
        // Lösung sieht gut aus; untere Schranke ungleich oberer Schranke,
        // untere Schranke nicht kleiner als globale untere Schranke
        // daher zu Menge der zu bearbeitenden Probleme hinzufügen
        P1 = P; P1[P1[0]] = 0; P1[0]++;
        P2 = P; P2[P2[0]] = 0; P2[0]++;
        X = X + {P1}; X = X + {P2};
      }
    }
  }
}
```

Für die Darstellung der Mengenoperationen Vereinigung und Mengendifferenz gelten dieselben Konventionen wie bei Beispiel 6.3.

Der angegebene Algorithmus läuft wie folgt ab:

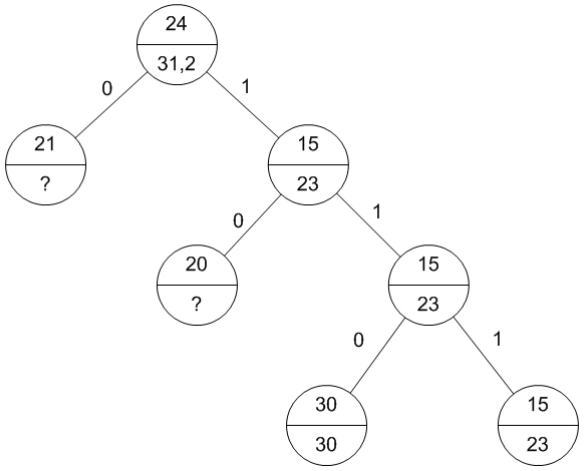
- Für die fünf Algorithmen wird der Nutzen berechnet
  - Algorithmus A (1): $7/40 = 0,715$
  - Algorithmus B (2): $3/10 = 0,3$

  - Algorithmus C (3): $1/6 \approx 0,16$
  - Algorithmus D (4): $1/5 = 0,2$
  - Algorithmus E (5): $13/90 \approx 0,14$

- Die Algorithmen werden nach Nutzen sortiert; man erhält die Reihenfolge $\{2, 4, 1, 3, 5\}$.
- Die untere Schranke wird berechnet: Die Algorithmen 1, 2, 3 und 4 werden ausgewählt; Zeit: 130 Minuten, Punkte (untere Schranke): 24, neue globale untere Schranke 24.
- Die obere Schranke wird berechnet: 1, 2, 3 und 4 werden ausgewählt; restliche Zeit ist $r = 50$, die Schranke ergibt sich als $w + r \cdot \frac{p_5}{ti} = 24 + 50 \cdot \frac{13}{90} \approx 31,22$.
- Algorithmus 2 wird fixiert (nicht hinzugenommen): Als untere Schranke ergibt sich nun eine Punkteanzahl von 21 bei Auswahl der Algorithmen 1, 3 und 4 sowie einer Zeit von 120 Minuten. Diese Lösung ist schlechter als die beste bereits erhaltene, und kann daher verworfen.
- Algorithmus 2 wird also fixiert (hinzugenommen): Es ergibt sich die Elternlösung.
- Des weiteren wird Algorithmus 4 fixiert (nicht hinzugenommen): Als untere Schranke ergibt sich nun eine Punkteanzahl von 20 bei Auswahl der Algorithmen 1, 2 und 3 sowie einer Zeit von 110 Minuten. Auch diese Lösung kann verworfen werden, da sie schlechter ist als die beste bereits erhaltene.
- Also wird Algorithmus 4 hinzugenommen. Es ergibt sich die Elternlösung.
- Nun wird Algorithmus 1 fixiert (nicht hinzugenommen): Bei Auswahl der Algorithmen $2, 3, 4, 5$ ergibt sich eine untere Schranke von 30 Punkten bei einer Zeit von 180 Minuten; die neue globale untere Schranke ist somit 30.
- Als obere Schranke ergibt sich ebenfalls 30, da alle Gegenstände, die noch nicht fixieren, problemlos in den Rucksack passen. Nun ist die untere Schranke gleich der oberen Schranke, also kann dieser Ast als abgeschlossen betrachtet werden.
- Zuletzt wird der Fall betrachtet, dass der Algorithmus fixiert wird als hinzugenommen: In diesem Fall ergibt sich wieder die Elternlösung, und damit eine Lösung, die schlechter ist als die beste bereits gefundene.

Insgesamt ergibt sich also als beste Lösung $2, 3, 4, 5$, was in den ursprünglichen Bezeichnungen B, C, D, E entspricht. Gearbeitet werden kann 180 Minuten, also die vollen zur Verfügung stehenden drei Stunden, bei einer Anzahl von 30 höchstens erreichbaren Punkten.

Der entsprechende Branch-and-Bound-Baum sieht wie folgt aus:



Die obere Zahl in jedem Knoten gibt jeweils die untere Schranke an, die untere Zahl die obere Schranke. Steht als obere Schranke ein Fragezeichen (?), so ist diese Schranke irrelevant, da diese Lösung bereits auf Grund der unteren Schranke aussortiert werden kann.

Aufgabe 6.5

Aufgabenstellung:

Eine Instanz des *Subset-Sum-Problems* besteht aus einer Menge A von n ganzen positiven Zahlen und einer oberen Schranke T. Wir bezeichnen eine Teilmenge A' von A als zulässig, wenn die Summe aller Elemente von A' höchstens T ist.
Die Summe der Elemente in einer zulässigen Teilmenge bezeichnen wir auch als den Wert der Teilmenge. Wir suchen eine zulässige Teilmenge von A mit einem möglichst großen Wert (im Idealfall T).
Im Folgenden nehmen wir an, dass für ein gegebenes Subset-Sum-Problem eine zulässige Teilmenge von A mit Wert größer oder gleich $\frac{T}{2}$ existiert.

(a) Geben Sie Pseudo-Code für einen polynomiellen Algorithmus an, der eine zulässige Teilmenge mit einem Wert von mindestens $\frac{T}{2}$ berechnet.

(b) Geben Sie die Laufzeit Ihres Algorithmus in Θ-Notation an und begründen Sie diese Laufzeit.

(c) Beweisen Sie, dass Ihr Algorithmus eine Lösung mit einem Wert von mindestens $\frac{T}{2}$ berechnet.

Lösung:

(a) Auch hier handelt es sich wieder um nichts anderes als um ein Rucksackproblem: Es wird für alle „Gegenstände“ probiert, ob sie noch in den Rucksack passen, wenn ja, werden sie eingepackt:

```
SubsetSum(A, n, T) {
  Sortiere A absteigend;
  L = {};
  sum = 0;
  for(i=1; i<=n; i++) {
    if(sum + A[i]<= T) {
      L = L + A[i];
      sum += A[i];
    }
  }
  return L;
}
```

$L + A[i]$ steht wieder für die Mengenoperation $L \cup A[i]$, wohingehen $sum+ = A[i]$ eine Rechenoperation mit Zahlen darstellt.

(b) Die Sortierung liegt in $\Theta(n \cdot \log(n)$, die Schleife liegt in $\Theta(n)$, insgesamt ergibt sich also eine Laufzeit in $\Theta(n \cdot \log(n)$.

(c) Beweis indirekt: Angenommen, es gibt eine Menge A sowie einen Wert T, sodass der Algorithmus keine gültige Lösung findet. Das bedeutet allerdings, dass

  (a) es kein Element $a \in A$ mit $\frac{T}{2} < a < T$ gibt: Sei B Teilmenge von A, wobei für alle Elemente $b \in B$ gilt: $\frac{T}{2} < b < T$. Dann gibt es auch ein Element $b \in B$, sodass $b \geq c$ für alle $c \in B$ gilt (das heißt b ist das größte Element von B). Dieses Element wäre jedoch das erste, das vom vorliegenden Algorithmus ausgewählt und in die Menge L aufgenommen würde. Also muss B eine leere Menge sein, d. h. es gibt kein Element $a \in A$ $\frac{T}{2} < a < T$.

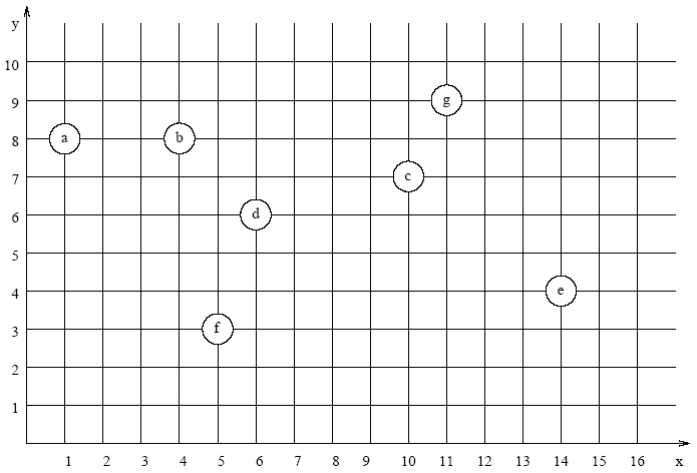
  (b) die Summe aller Elemente $a_1, a_2, \ldots, a_p \in A$, *peqn* mit $a_i < \frac{T}{2}$, $1 \leq i \leq p$ kleiner als $\frac{T}{2}$ sein muss. Wäre diese Summe größer als $\frac{T}{2}$, so hätte der Algorithmus die entsprechenden Elemente $a_i \in A$ gefunden, sodass $\sum a_i > \frac{T}{2}$ ist.

D.h. auch die Summe aller Elemente $a_1, a_2, \ldots, a_p \in A$, $p \leq n$ mit $a_i < T$, $1 \leq i \leq p$ ist kleiner als $\frac{T}{2}$. Das ist aber ein Widerspruch zur Voraussetzung der Angabe, dass zumindest eine Lösung des Subset Sum-Problems existiert.

Aufgabe 6.6

Aufgabenstellung:

Gegeben sei der aus den nachstehend abgebildeten Knoten aufgebaute vollständige, ungerichtete Graph G.



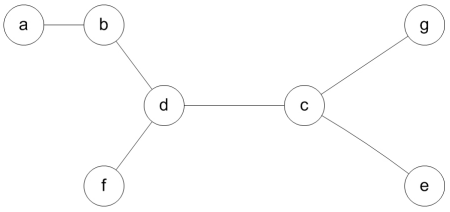
Für einen Knoten i bezeichne $x(i)$ und $y(i)$ dessen Koordinaten auf dem Gitter. Das Gewicht einer Kante $e = (i, j)$ in G sei definiert als die Manhattan-Distanz

$$M(i, j) = |x(i) - x(j)| + |y(i) - y(j)|$$

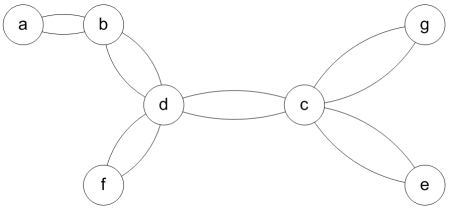
der Knoten i und j.
Wenden Sie die *Spanning-Tree-Heuristik* für das Traveling-Salesman-Problem an, um im so definierten Graphen eine Tour zu finden. Visualisieren und beschreiben Sie alle Schritte. Wählen Sie den Knoten a als Startpunkt.

Lösung:

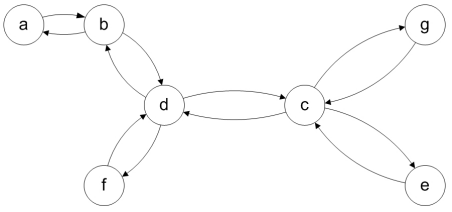
1. Ermitteln des MST mit Hilfe des Algorithmus nach Kruskal (alternativ ist auch Prim möglich); es werden nacheinander die Kanten (a, b), (c, g), (f, d), (b, d), (c, d) und (c, e) ausgewählt um einen MST zu erhalten:



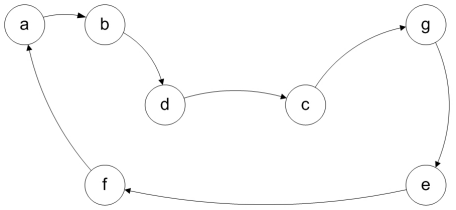
2. Verdopplung aller Kanten:



3. Ermitteln einer Eulertour, ausgehend vom Knoten a:



4. Ermitteln der Rundtour:

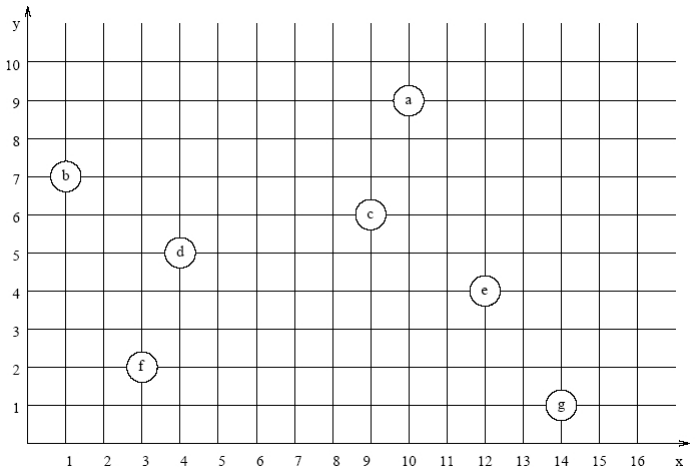


Die Länge dieser Rundtour beträgt 42.

Aufgabe 6.7

Aufgabenstellung:

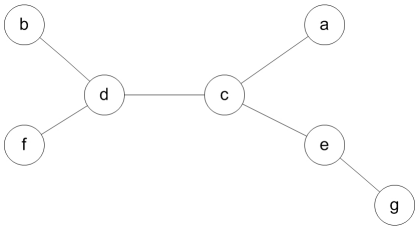
Gegeben sei der aus den nachstehend abgebildeten Knoten aufgebaute vollständige, ungerichtete Graph G.



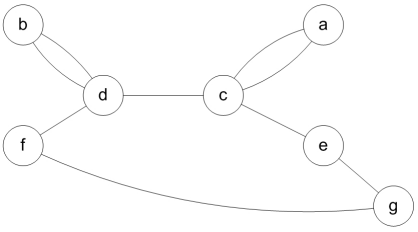
Das Gewicht jeder Kante entspricht jeweils der euklidischen Distanz zwischen den beiden Knoten, die sie verbindet. Wenden Sie die Christophides-Heuristik an, um einen möglichst kurzen Rundweg in diesem Graphen zu finden. Beschreiben und veranschaulichen Sie dabei alle Schritte. Wählen Sie den Knoten a als Startpunkt.

Lösung:

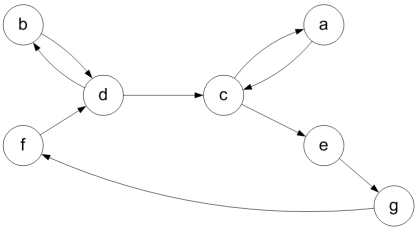
1. Ermitteln des MST mit Hilfe des Algorithmus von Kruskal (alternativ kann auch der Algorithmus von Prim eingesetzt werden); nacheinander werden die Kanten (d, f), (a, c), (b, d), (c, e), (e, g) sowie (c, d) ausgewählt:



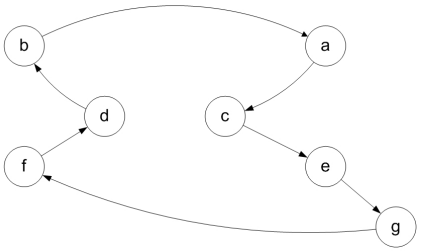
2. Bestimmung eines perfekten Matchings kleinsten Gewichts in dem von der Menge der Knoten mit ungeradem Grad induzierten Untergraphen:



3. Ermitteln der Eulertour:



4. Ermitteln der Rundtour:



Die Gesamtlänge der Rundtour beträgt

$$\sqrt{10} + \sqrt{13} + \sqrt{13} + \sqrt{122} + \sqrt{10} + \sqrt{13} + \sqrt{85} \approx 33,80$$

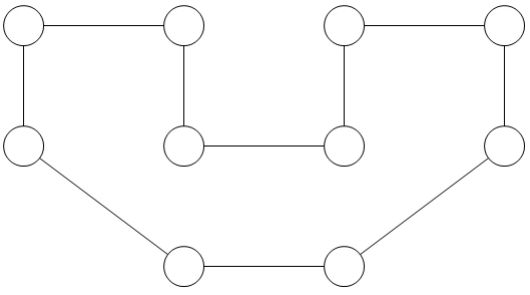
Aufgabe 6.8

Aufgabenstellung:

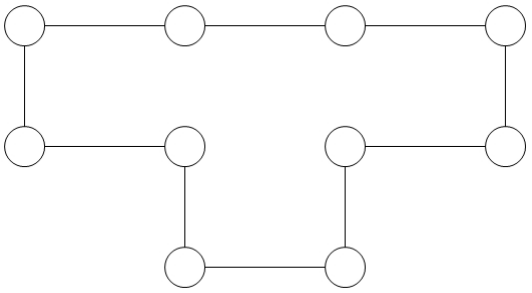
Überlegen Sie sich zum Euklidischen Symmetrischen TSP (Traveling Salesman Problem) ein Beispiel, in dem die Verbesserungsheuristik 2-OPT eine Tour nicht weiter verbessern kann, obwohl diese noch nicht global optimal ist.

Lösung:

Die folgende Lösung für ein TSP kann nicht mittels 2-OPT verbessert werden:



Die bessere Lösung wäre nämlich:



Aufgabe 6.9

Aufgabenstellung:

Analysieren Sie die Laufzeit einer Iteration von r-OPT für das Symmetrische TSP auf einem vollständigen Graphen mit N Knoten:

(a) Wie groß ist die Menge Z?

(b) Bis zu wie viele unterschiedliche Möglichkeiten kann es geben, um nach dem Entfernen von r Kanten die entstandenen r Pfade wieder zu einer gültigen Tour zu verbinden?

Geben Sie eine daraus resultierende obere Schranke der Laufzeit einer Iteration (eines möglichen Verbesserungsschritts) von r-OPT in O-Notation an.

Lösung:

(a) $|Z| = \frac{|T|!}{(|T|-r)!\cdot r!} = \binom{|T|}{r}$

(b) Die ursprüngliche Tour zerfällt durch Entfernen

- einer Kante in eine Komponente,
- zweier Kanten in zwei Kompontenten
  ...
- von r Kanten in r Komponenten.

Zum Zusammenfügen gibt es nun folgende Möglichkeiten:

- Bei einer Komponente gibt es eine Möglichkeit des Zusammenfügens,
- bei zwei Komponenten gibt es zwei Möglichkeiten,
- bei drei Komponenten gibt es $3 \cdot 2 = 3! = 6$ Möglichkeiten,
  ...
- bei r Komponenten gibt es $r!$ Möglichkeiten.

D. h. in einem Verbesserungsschritt werden alle r-elementigen Teilmengen von T bestimmt, und anschließend werden daraus neue Touren gebildet. Es ergibt sich damit insgesamt folgende Laufzeit:

$$r! \cdot \frac{|T|!}{(|T|-r)!\cdot r!} = \frac{|T|!}{(|T|-r)!} = \underbrace{|T| \cdot (|T|-1) \cdot (|T|-2) \cdot \ldots \cdot (|T|-r+1)}_{r \text{ Faktoren}} = O(|T|^r)$$

Aufgabe 6.10

Aufgabenstellung:

Sie wollen mit Hilfe von *Simulated Annealing* in einem ungerichteten, gewichteten, zusammenhängenden Graphen $G = (V, E)$ einen möglichst langen Pfad finden, der jeden Knoten des Graphen höchstens einmal besucht. Dazu brauchen Sie eine Nachbarschaftsfunktion, die aus einer gültigen Lösung eine neue gültige Lösung erzeugt.

(a) Geben Sie eine geeignete Nachbarschaftsfunktion in Pseudocode an, welche die notwendige Bedingung *Erreichbarkeit des Optimums* erfüllt.

(b) Zeigen Sie, dass Ihre Funktion bei Eingabe eines Pfades stets wieder einen Pfad produziert, der sich vom Eingabe-Pfad unterscheidet und jeden Knoten von G höchstens einmal besucht.

(c) Sei p ein beliebiger Pfad in G. Zeigen Sie, dass man durch mehrmaliges Anwenden Ihrer Nachbarschaftsfunktion aus p einen Pfad maximaler Länge in G konstruieren kann.

Lösung: