Informatik I WS 02/03

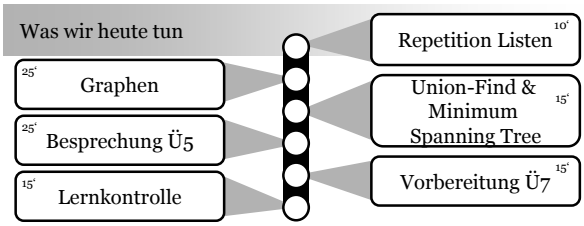
17. Dezember 2002

Übungsserie 6 abgeben

Buchtip – müsst Ihr im nächsten Semester sowieso kaufen

Algorithmen und Datenstrukturen
T. Ottmann / P. Wittmayer
Spektrum Verlag, ISBN 3-8274-0110-0

Was wir heute tun



Arrays versus Listen

Array:

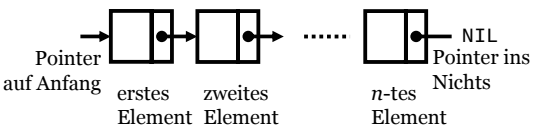
- fixe Länge
- beschränkte Länge
+ direkter Zugriff

„Listen“:

+ variable Länge, Einfügen & Löschen möglich
+ annähernd unlimitierte Länge
- kein direkter Zugriff

Listen

Liste besteht aus Anfang, Elementen & Ende



ein Element:

Teil mit Nutzdaten

Pointer auf Nachfolger

Ein Listenelement in Oberon

Typdeklaration in Oberon:

```
TYPE Element = POINTER TO RECORD
     wert : INTEGER;
     naechster : Element;
END;
```

Wieder ein Record das selben Typ hat, weil Nächster auch ein Element ist

Element ist
Pointer auf ein Record mit
einem Teil mit Nutzdaten (`wert`) und
einem Pointer auf Nachfolger
(`naechster` mit Typ `Element`!)

Operationen auf Listen

Elemente einfügen in Listen?

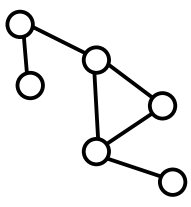
Elemente aus Listen entfernen?

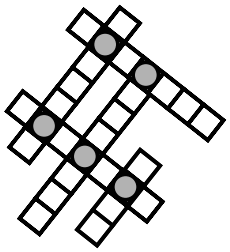
Graphen

eine Möglichkeit:
Graph ≈ mehrere verwobene Listen

Verbindung —— Liste

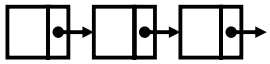
Knoten ○ Schnittpunkt/Kreuzung ●



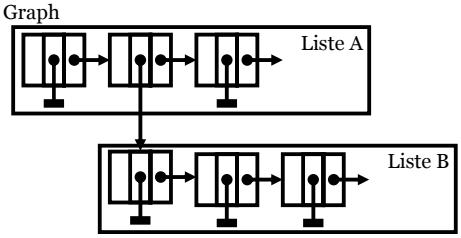


Implementation von Graphen

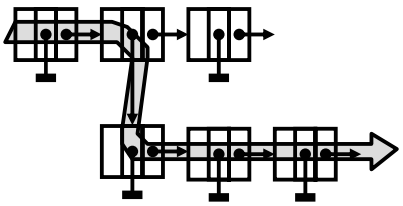
Liste



Graph



Traversierung von Graphen



pro Knoten zwei Pfeile, zwei weitere Wege

Methoden, um Graphenprobleme zu lösen:
- Backtracking-Strategie
- Union-Find-Struktur

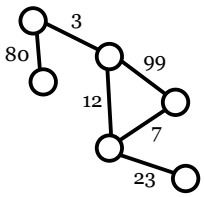
Union-Find-Struktur

Struktur verlangt folgende 3 Funktionen:
- *MakeSet(e)*
  erstelle neue Menge mit einzigem Element e
- *Find(x)*
  liefert die Menge, die Element x enthält
- *Union(e, f)*
  vereinigt die Mengen e und f zu neuer Menge

wie wird das implementiert?

- jede Menge ist eine Liste:
  MakeSet erstellt neue Liste,
  Find gibt erstes Element der enthaltenden Liste,
  Union verknüpft zwei Listen
- jede Menge ist ein Baum

Minimum Spanning Tree - minimaler spannender Baum



Graphen mit
Gewichten auf den Kanten

„Ortschafen mit Strassenverbindungen; die Gewichte geben die Länge der Strassen an“

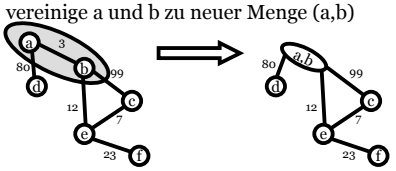
ein minimaler spannender Baum T für Graph G besteht aus allen Knoten V von G, enthält aber nur eine Teilmenge E‘ der Kantenmenge E von G, die alle Knoten des Graphen miteinander verbindet und die Eigenschaft hat, dass die Summe aller Kantengewichte den minimal möglichen Wert hat unter allen Teilmengen von E, die alle Knoten des Graphen G miteinander verbindet.

Minimum Spanning Tree finden

Kruskals Verfahren aus dem Jahre 1956

❶ sortiere Kanten in einer Liste nach Gewicht

❷ nimm kleinste Kante &
vereinige a und b zu neuer Menge (a,b)



❸
- wiederhole das mit allen Kanten, bei welchen zwei Mengen jeweils vereint werden können
- stoppe, wenn alle Knoten des Graphen verbunden sind

Kruskals Verfahren in Pseudocode

```
PROCEDURE Kruskal( g : GRAPH, VAR mst : LISTE )
    VAR kantenliste, resultat: LISTE;
BEGIN
    mst := leer;
    sortiere alle Kanten aus g in Liste kantenliste;
    FOR alle Knoten DO
        MakeSet( Knoten );
    END;
    WHILE noch mehrere Mengen vorhanden DO
        nimm kleinste Kante (v, w);
        entferne Kante aus kantenliste;
        IF Find(v) # Find(w) THEN
            Union( Find(v) , Find(w) );
            mst := mst & (v, w);
        END
    END
END Kruskal;
```

Besprechung Übungsserie 5 Aufgabe 1

Konversion von ARRAY OF CHAR in LONGINT

```
MODULE Decoder;
IMPORT In, Out;
PROCEDURE ToInt(x: ARRAY OF CHAR) : LONGINT;
  VAR result, i, digit: LONGINT;
BEGIN
  result := 0; i := 0;
  WHILE x[i] # 0X DO
    IF (x[i] < "0") OR (x[i] > "9") THEN
      RETURN -1
    END;
    digit := ORD(x[i]) - ORD("0");
    IF (result > (MAX(LONGINT) - digit) DIV 10) THEN
      RETURN -2
    END;
    result := result * 10 + digit;
    INC(i);
  END;
  RETURN result;
END ToInt;
END Decoder;
```

Eingabe

Ausgabe

Eingabe ausserhalb Definitionsbereich

Fehlercode

berechnete Ausgabe übersteigt Format des LONGINT

letztes Resultat rutscht mathematisch nach links

Besprechung Übungsserie 5 Aufgabe 3 – Teil 1

Schnelle Erkennung von Schlüsselwörtern

```
MODULE Sherlock;
 IMPORT In, Out, Scanner;
 CONST   MaxHash = 1000;
         MaxWord = 50;
         MaxS = 100;
 TYPE    String = ARRAY MaxWord OF CHAR;
 VAR     Schluesselworte: ARRAY maxSl OF String;
         Hashtab: ARRAY MaxHash OF String;
         P: INTEGER;

 PROCEDURE InitHashtab(len: INTEGER);
 PROCEDURE H(s: String): INTEGER;
 PROCEDURE Register*();
 PROCEDURE Search*();
END Sherlock.
```

Besprechung Übungsserie 5 Aufgabe 3 – Teil 2

Initialisieren der Hashtabelle

```
PROCEDURE InitHashtab(len: INTEGER);
  VAR i: INTEGER;
BEGIN FOR i:=0 TO len DO
  Hashtab[i] := 0X;
  END;
END InitHashtab;
```

Besprechung Übungsserie 5 Aufgabe 3 – Teil 3

die Hashfunktion

```
PROCEDURE H(s: String): INTEGER;
   VAR i, res: INTEGER;
BEGIN
   i:=0;
   WHILE (s[i] # 0X) DO
      res := res + (ORD(s[i]) * 17 * (i+1));
      INC(i);
   END;
   res := res MOD P;
   RETURN res;
END H;
```

Besprechung Übungsserie 5 Aufgabe 3 – Teil 4

ein Wort registrieren

```
PROCEDURE Register*();
  VAR   anzSchluessel, i,
        currenthash: INTEGER;
        collision: BOOLEAN;
BEGIN
  In.Open;
  WHILE ~In.Done DO
    In.String( Schluesselwoerter[anzSchluessel] );
    INC(anzSchluessel);
  END;
  DEC(anzSchluessel);
  P := anzSchluessel;
  collision := TRUE;

...
```

Besprechung Übungsserie 5 Aufgabe 3 – Teil 5

ein Wort registrieren

```
WHILE collision DO
  i:=0;
    LOOP
      currenthash := H(Schluesselwoerter[i]);
      IF( Hashtab[currenthash] = 0X ) &
      ( i < anzSchluessel) THEN
        Hashtab[currenthash]:= Schluesselwoerter[i];
      ELSIF i = anzSchluessel THEN
        collision := FALSE; EXIT;
      ELSE
        InitHashtab(P);
        INC(P);
        EXIT;
      END;
      INC(i);
    END;
END; END Register;
```

Besprechung Übungsserie 5 Aufgabe 3 – Teil 6

ein Wort registrieren

```
PROCEDURE Search*();
  VAR   filename: String;
        current: String;
BEGIN
  In.Open; In.String(filename);
  Scanner.Open(filename);
  WHILE ~Scanner.Done DO
    Scanner.GetWord(current);
    IF Hashtab[H(current)] = current THEN
      Out.String(current);
      Out.Ln;
    END;
  END;
END Search;
```

Vorbereitung Übungsserie 7 Aufgabe 1

Minimum Spanning Tree

Kruskals Verfahren
`http://www-b2.is.tokushima-u.ac.jp/~ikeda/suuri/kruskal/Kruskal.shtml`

Prim Algorithmus – evtl. einfacher:
`http://www.iti.fh-flensburg.de/lang/algorithmen/graph/spanning.htm`

empfohlenes Vorgehen:
1. Union-Find Modul erstellen
2. Kruskals Verfahren implementieren

Vorbereitung Übungsserie 7 Aufgabe 2

elektronischer Fahrplan

EBNF der Daten:
```
Netz = {Linie}.
Linie = Liniennummer Stationsname
        { Fahrzeit Stationsname } „-1“.
```

10	...	4	Milchbuck	2	Irchel	3	...	-1

1. also einfach „durchRIDERN“ und in Liste einlesen
2. gleiche Haltestellen mit einer Kante mit 4 verbinden