A comparative study of alternative Haskell implementations of

The Maximum Matching Problem

Zixiao HE (Jake)
31 May 2004

Goals

- Examine a functional approach to graph algorithms.
- Monadic vs. Non-monadic implementations
- Investigate the benefits of implementing graph algorithms using a higher order language.

Maximum Cardinality Matching

The graph algorithm selected for this study is an classic problem in Matching Theory of Graph Theory.

Applications:

- Vehicle routing problems ('Chinese postman' problem)
- Determination of shortest chain in a graph
- Assignment of jobs

The Battle of Britain

During the Battle of Britain some British air squadrons consisted of planes with two pilots. However, certain pairs of pilots could not fly together because of differences of language or of habits. It was then required to find the largest number of planes which could fly at the same time.

This reduces to finding a maximum cardinality matching in the graph whose vertices represent the pilots, where two vertices are linked by an edge if and only if the corresponding pilots could make up a team.

Definitions

- **Matching**
  Given a simple non-directed graph ***G = [V,E]*** (***U*** is a set of edges), a matching is a subset of edges ***M*** subset of ***E*** such that no two edges of ***M*** are adjacent.

  A vertex ***v*** is matched if it is in some edge of the matching; otherwise ***v*** is unmatched.

- **Maximum Matching**
  M is a maximum matching if no matching on ***G*** contains more edges than ***M***. See Example.

More Definitions

- **Alternating Path**
  An alternating path in a matched graph is a ***path (v1,...,vn)*** such that exactly one of every two edges ***vi-1vi*** and ***vivi+1*** is matched, for ***1 < i <n***.

- **Augmenting Path**
  An augmenting path is an alternating path whose ends v1 and vn are distinct unmatched vertices. See Example.

Edmond's Algorithm using Gabow's Labelling System

The original Edmond's algorithm is not complex, nor is it trivial.

The intrinsic idea is this:

- Algorithm ***E*** begins with all vertices unmatched. It searches for an augmenting path. If such a path is found, the matching is augmented. The new matching contains one more edge than the previous one. Next ***E*** searches for an augmenting path for the new matching. This process is iterated.

- Eventually, ***E*** constructs a matching that has no augmenting path. This matching is maximum, by ***Berge's Lemma***.

Search for an Augmenting Path

Algorithm E searches for an augmenting path in the following way.

- First an unmatched vertex ***u*** is chosen. ***E*** scans edges to find alternating paths to ***u***. A vertex is called ***outer*** when ***E*** finds an alternating path from ***v*** to ***u*** that starts with a matched edge. Let such a path be ***P(v) = (v, vl, ... , u)***, so ***vvl*** is matched. ***E*** sets an entry in the ***LABEL*** array for every ***outer*** vertex ***v***. Path ***P(v)*** can be computed from ***LABEL(v)***. If an edge joining an outer vertex ***v*** to an unmatched vertex ***u' ~ u*** is scanned, ***E*** finds an augmenting path, ***(u').P(v) = (u', v, vl, . . . , u)***. If no such edge is ever scanned, vertex ***u*** is not in an augmenting path.[3]

L

L assigns the edge label $n(xy)$ to nonouter vertices. Edge xy joins outer vertices x, y. *L* sets *join* to the first nonouter vertex in both $P(x)$ and $P(y)$. Then it labels all nonouter vertices preceding *join* in $P(x)$ or $P(y)$. See Figure 4.

R

$R(v, w)$ rematches edges in the augmenting path. Vertex v is outer. Part of path $(w) \bullet P(v)$ is in the augmenting path. It gets rematched by $R(v, w)$ (Although *R* sets $MATE(v) \leftarrow w$, it does not set $MATE(w) \leftarrow v$. This is done in step E3 or another call to *R*.) *R* is a recursive routine.

States

- Monadic STRef
- Non-Monadic pass by argument

Date Structure

- FiniteMap Sufficient functions

The Haskell implementation is more focused on correctness rather then efficiency.

Problems

Ordered in increasing magnitude of hours spent and headache.

- Maintaining States
- Debugging
- An functional approach to a low level description of a non-trivial graph algorithm.

Concluding Remarks

It is rather an expression of the intuitiveness of the algorithm.
So much easier to write

(given that you know the intuitiveness of the of the algorithm)

- reduce errors
- code elegance
- code readability

QUESTIONS ?