# L19-20 Bi-partite Matching & Revision Lecture

## G54AAD Advanced Algorithms and Data Structures

Per Kristian Lehre

December 4th, 2015

# Matching

## Definition (Matching)

A **matching** of a graph $G = (V, E)$ is a subset $M \subset E$ of its edges such that for all distinct edges $\{u, v\} \in M$ and $\{x, y\} \in M$

- $\{u, v\} \cap \{x, y\} = \emptyset$

i.e., no two edges in $M$ share a common vertex.

## Example

# Bi-Partite Graph

## Definition (Bipartite Graph)

A graph $G = (V, E)$ is called **bi-partite** if there exists $S, T \subseteq V$ st.

- $S \cup T = V$ and $S \cap T = \emptyset$ (i.e., $S$ and $T$ is a partition of $V$), and
- for all $\{u, v\} \in E$, either
  - $u \in S$ and $v \in T$, or
  - $u \in T$ and $v \in S$

## Example

# Maximum Bi-partite Matching Problem

**Problem**

Given a bi-partite graph, find a matching with maximal cardinality.

**Example (Allocating gates to flights)**

Allocate gates to as many flights as possible within a time window.
Flights have gate constraints, e.g.

- some types of aircraft require certain gates
- airlines may have certain requirements

# Solving Maximum Bi-partite Maching as Max Flow

Converting a bi-partite graph to a flow network

1. Add an edge from $s$ to all nodes in $S$
2. Add an edge from $t$ to all nodes in $T$
3. Give all edges capacity 1.

# Solving Maximum Bi-partite Maching as Max Flow

Converting a bi-partite graph to a flow network

1. Add an edge from $s$ to all nodes in $S$
2. Add an edge from $t$ to all nodes in $T$
3. Give all edges capacity 1.

**Theorem (Integrality Theorem)**

*If all capacities in a flow network are integer-valued, then the Ford-Fulkerson method produces a maximum flow $f$ where $f(u, v)$ is integer-valued for all pairs $(u, v)$.*

**Theorem**

*The bi-partite graph has a matching of size $k$ if and only if the flow network has a flow of value $k$.*

# Outline of revision lecture

- Revision of some of the topics covered
- Example questions from each topic
- Revision material (past exam papers in G64ADS)
- Complete syllabus given on Moodle
  (see reading column and lecture notes)

# Assessment

| Activity | % of Overall Mark |
|---|---|
| Exam (2 hr written) | 100 % |
| Coursework (report) | 25 % |

# Exam Format[1]

G54AAD-E1

**The University of Nottingham**

SCHOOL OF COMPUTER SCIENCE

A LEVEL 4 MODULE, AUTUMN SEMESTER 2015-2016

**G54AAD ADVANCED ALGORITHMS AND DATA STRUCTURES**

Time allowed TWO hours

---

*Candidates may complete the front cover of their answer book and sign their desk card but must NOT write anything else until the start of the examination period is announced.*

***Answer Question 1 in Section A and TWO questions in Section B.***

*No calculators are permitted in this examination.*

*Dictionaries are not allowed with one exception. Those whose first language is not English may use a standard translation dictionary to translate between that language and English provided that neither language is the subject of this examination. Subject-specific translation directories are not permitted.*

[1] Subject to suggestions by external moderator.

# General advice

- Format
  - Time allowed is 2 hours.
  - Section A (40%): Answer everything
  - Section B (60%): Answer two of three questions.
- Read the entire paper before answering (plan your time).
- Read questions carefully.
- Answers have to be justified.
- State any needed assumptions.

# Basics assumed

- Solving equations, inequalities
- Proof by induction, contradiction etc.
- Properties of basic functions,
  eg logarithms, exponential function, factorials etc.

# L01-02 Asymptotic Notation & Recurrence relations

Asymptotic notation

- Definitions

**Example questions**

- Explicit: Classify functions according to asymptotic growth rate.
- Implicit: State answers to other problems in asymptotic notation.

# L03-04 The Master Theorem

Recurrences and the Master theorem

- Statement
- Proof (disregarding round-off errors)
- Applicability (when can the theorem be used)

**Example questions**

- Explicit: Solve some given recurrences.
- Implicit: Runtime analysis of a divide and conquer algorithm, where the runtime can be expressed as a recurrence for which the theorem applies.

# L05-06 Leftist Heaps

```haskell
merge :: (Ord a) => LHeap a -> LHeap a -> LHeap a
merge x E = x
merge E y = y
merge h1@(B _ x1 left1 right1) h2@(B _ x2 _ _)
    | x1 <= x2   = B (1 + (rank right)) x1 left right
    | otherwise  = merge h2 h1
    where (left,right) = leftistOrder left1 (merge right1 h2)

leftistOrder :: LHeap a -> LHeap a -> (LHeap a, LHeap a)
leftistOrder h1 h2
    | (rank h1) <= (rank h2) = (h2,h1)
    | otherwise              = (h1,h2)
```

- Definition of heaps, min-heaps, max-heaps.
- Definition of leftist heaps, including leftist heap property
- Analysis of height of leftist heaps
- Implementation and analysis of operations (findMin, merge, insert, delMin)

**Example question**

- Explicit: Implementation of leftist heaps.
- Implicit: Use some of the ideas behind leftist heaps to analyse other data structures.

# L07-08 Binomial Heaps

Binomial trees

- Definition
- Properties: eg rank vs size or depth

Binomial heaps

- Definition
- Properties: eg largest rank vs size etc.
- Implementation and analysis of basic operations insTree, merge, findMin, deleteMin

**Example question:**

- Find size, depth, and number of children in a binomial tree of rank r.

# L09 Persistence

Red Black Trees

- Persistent vs ephemeral data structures
- Sharing of data

**Example question**

- Illustrate sharing of data from example code.

# L10 Red-Black Trees

Red Black Trees

- Definition and properties
- Implementation and analysis of operations
  eg member, insert (delete not required)

**Example question**

- Analysis of depth.

# L11-12 Dynamic Programming

Dynamic Programming

- Applicable on problems with
  - optimal substructure
  - overlapping substructure
- Longest Common Subsequence
- Floyd-Warshall Algorithm

**Example question**

- Explicit: Define solution to problem recursively in terms of sub-solutions.
- Implicit: Recognise that a problem can be solved by dynamic programming.

# L12-18 Maximum Flow

Maximum flow

- Definition and properties of flows
- Augmenting paths and Ford-Fulkerson method
- Cuts and Max-flow Min-cut Theorem
- Proof ideas

**Example questions**

- make use of basic properties of flows
- find residual networks and augmenting paths
- recognise that a problem is an instance of max-flow

# L19-20 Edmonds-Karp Algorithm

Edmonds-Karp

- Instance of Ford-Fulkerson method
- Augmenting paths by Shortest paths
- Analysis of runtime

**Example question**

- write pseudo-code
- explain behind runtime analysis

# L21 Bi-partite Matching

Bi-partite matching

- problem definition
- formulation as max-flow

**Example question**

- Show that some real-world problem can be described as bipartite matching.

# Revision Material on Moodle

- G54AAD was a new module from 2014/2015, but overlaps with old module G64ADS.
- Three past exam papers available on Moodle
  - G64ADS regular exam 2012/2013
  - G64ADS regular exam 2013/2014
  - G54AAD regular exam 2014/2015
- Relevant exam questions from G64ADS are marked in red.

# Thank you!