Advanced Algorithmic Techniques (COMP523)

Module Recap

Lectures 1-5

- We designed and analysed algorithms for *searching*, *sorting*, *majority*, *selection*, *closest pair of points*, *integer multiplication*.
- Emphasis on: *correctness*, *running time*, *memory*.
- The design paradigm here was *divide-and-conquer*.

Lectures 6-8

- We designed and analysed *graph algorithms* for *graph traversal (DFS, BFS)*, *bipartiteness*, *strong connectivity*, *topological ordering*, *strongly connected components.*
- Emphasis on: *correctness*, *running time*.
- Additional emphasis on: *graph concepts, which were used in other parts of the module*.
- The design paradigm here was *divide-and-conquer*.

Lectures 9-11

- We designed and analysed *greedy algorithms* for *interval scheduling*, *minimum spanning trees*, *clustering.*
- Emphasis on: *correctness.*
- Additional emphasis on: *the greedy paradigm was used in other parts of the module.*
- The design paradigm here was *greedy*.

Lectures 12-13

- We designed and analysed *dynamic programming algorithms* for *weighted interval scheduling*, *subset sum* and *knapsack.*
- Emphasis on: *correctness, running time (polynomial time vs pseudo-polynomial time).*
- The design paradigm here was *dynamic programming*.

Lectures 14-16

- We designed and analysed *network flow algorithms* for *maximum flow* and *minimum cut.*
- Emphasis on: *correctness, running time,*
- Additional emphasis on: *modelling with flows, how to use flow algorithms to solve other problems.*
- The design paradigm here was *greedy*.

Lectures 17-18

- We discussed the notions of NP-hardness and NP-completeness and their implications for the polynomial-time solvability of problems.

- We discussed the concept of a *reduction*, which is used to show NP-hardness.

- We saw a catalogue of NP-complete problems.

- Emphasis on: *The implications of NP-completeness, the decision vs optimisation versions of problems and the fact that we dealt with mostly NP-complete problems in the remainder of the module.*

Lecture 19

- We introduced the concept of linear programs and integer linear programs.
- We introduced the notion of duality.
- Emphasis on: *Modelling problems as LPs and ILPs to solve them either exactly or approximately (relevant to the following lectures). We also later used duality to design approximation algorithms for some problems.*
- Additional emphasis on: *Using total unimodularity to show that some LPs (e.g., flows) have integer solutions.*

Lectures 20-23

- We designed and analysed various *approximation algorithms* for NP-hard problems.
  - e.g., *load balancing*, *vertex cover*, *0/1-knapsack*.
- Emphasis on:
  - Notions: *Approximation ratio, inapproximability, integrality gap.*
  - Algorithms: *How to use the design techniques to come up with approximation algorithms and how to analyse their performance.*
- Design paradigms: *greedy*, *pricing method (primal-dual)*, *LP-relaxation and rounding*, *dynamic programming on rounded inputs*.

Lectures 24 - 27

- We designed and analysed *randomised algorithms* for various problems e.g., *minimum cuts*, *MAX-SAT* and *MAX-CUT*.

- Emphasis on:

  - *Working with probabilities (random variables, expectations, analysis).*

  - *Success Amplification.*

  - *Randomisation as a tool for approximation algorithms.*

- The design paradigms here were *greedy* and *LP-relaxation and rounding*.

Lectures 28 - 29

- We designed and analysed *online algorithms* for *online load balancing* and *paging*.
- Emphasis on:
  - *Notions: Competitive Ratio*
  - *Algorithms: Analysis for algorithms and impossibilities.*