CMPT231

Lecture 13: ch34

Tractability, Semester Review

1 Corinthians 1:18-21 (ESV)

For the word of the **cross** is **folly** to those who are
perishing,
but to us who are being saved it is the **power** of God.

19 For it is written, “I will destroy the **wisdom** of the wise,
and the **discernment** of the discerning I will thwart.”

20 Where is the one who is **wise**? Where is the **scribe**?
Where is the **debater** of this age?
Has not God made **foolish** the wisdom of the world?

Outline for today

- **All-pairs** shortest path
  - Johnson (reweighted Dijkstra): $O\left(|V|^2 \log|V| + |V||E|\right)$
- **Tractability**
  - Complexity **classes**: P, EXP, R
  - **Non-deterministic** verification: NP
  - NP-**hard** and NP-**complete**
- Semester **review**

All-pairs shortest path

- **Bellman-Ford** on each vertex: $O\left(|V|^2|E|\right)$
- Dyn prog by **path length**: $O\left(|V|^3\log|V|\right)$
- **Floyd-Warshall** (by subset of V): $O\left(|V|^3\right)$
- **Johnson** (Dijkstra on each vertex): $O\left(|V|^2\log|V| + |V||E|\right)$
  - **Reweight** edges for positive weights

Johnson's reweighting

- Johnson's trick: Add a **new** vertex s
  - Add **zero-weight** edges from s to all vertices:
    - $V' = V \cup \{s\}$,
    - $E' = E \cup \{(s,v) : v \in V\}$, and $w(s,v) = 0 \ \forall \ v$
  - **Compute** $y(s,v) \ \forall \ v$ (e.g., with Bellman-Ford)
  - **Reweight** edges using $h(v) = y(s,v)$

Johnson reweight, Fig-25-6(ab)

Johnson's reweighting

- Why does this eliminate **negative weights**?
    - **Triangle inequality**: $y(s,v) \le y(s,u) + w(u,v)$
    - Substitute **defn** of h: $h(v) \le h(u) + w(u,v)$
    - Apply **reweighting**: $w'(u,v) = w(u,v) + h(u) - h(v) \ge 0$

Johnson reweight, Fig-25-6(ab)

Johnson all-pairs

```
def Johnson( G, w ):
  create G' = (V', E') with new vertex s
  BellmanFord( G', w, s )

   # find dlt[ s, v ]: O(VE)
    if returned FALSE, quit

 # net-neg cycle
  w'[ u, v ] = w[ u, v ] + dlt[ s, u ] - dlt[ s, v ]
  for u in V:
    Dijkstra( G, w', u )

    # find dlt'[ u, v ]: O(VlgV + E)
    for v in V:
      d[ u, v ] = dlt'[ u, v ] - dlt[ s, u ] + dlt[ s, v ]
  return d
```

- Innermost loop **converts** back to original weighting
- **Complexity**: $O\left(|V|^2 \log|V| + |V||E|\right)$

All-pairs shortest path: summary

- **Negative** edges allowed (but not net-negative cycles)
- **Floyd-Warshall**: $O\left(|V|^3\right)$
  - Dyn prog by **subset** of vertices for intermediate nodes
- **Johnson**: $O\left(|V|^2 \log|V| + |V||E|\right)$
  - **Bellman Ford** from new source s
  - **Reweight**: h(v) = y(s,v) and w’(u,v) = w(u,v) + h(u) - h(v)
  - Run **Dijkstra** from each vertex

Outline for today

- All-pairs shortest path
  - Johnson (reweighted Dijkstra): $O\left(|V|^2 \log|V| + |V||E|\right)$
- **Tractability**
  - **Complexity classes:** P, EXP, R
  - Non-deterministic verification: NP
  - NP-hard and NP-complete
- Semester review

Decision problems

- Phrase problem as yes/no **decision**:
    - **Input**: a string s (e.g., encoded in binary)
    - **Problem**: a set of strings X
    - **Algorithm** A returns boolean:
      $A(s) = true \Leftrightarrow s \in X$
- **Complexity** of A in terms of **length** n of s
- e.g., given graph (V, E, w), is there a **path** of weight ≤ 4 between nodes u and v?
    - **Input** s includes entire graph spec, “4“, u, and v
    - Floyd-Warshall: $O\left(|V|^3\right) = O(n^3)$
- e.g., given an integer j, is it **prime**?
    - Writing j out, **length** of input is $n = \log j$

Turing machine model

- General model of **computation**
- Infinite **tape** using finite **symbol** set (plus blank)
- **Head** can read, write, and move left/right
- Machine has a finite **state space** and **transitions**:
  - Instructions for when to **change** internal states

![Alan Turing]
(static/img/alan-turing.jpg)

![Turing machine]
(static/img/turing-machine-2.png)

Complexity classes

- P: decision problems for which **polynomial-time** algorithms exist ("tractable")
  - **Most** of the algorithms in this course! $O(n^c)$
- EXP: problems solvable in **exponential** time: $O(2^{n^c})$
- R: problems solvable in **finite** time
  - R = "**recursive**" (Turing 1936, Church 1941)

P, EXP, and R

Examples

- Does a weighted graph have a **net-negative cycle**?
  - In P: e.g., Bellman-Ford $O(|V||E|) = O\left(n^2\right)$
- Given a **chess** board configuration (n x n), can White **win**?
  - In EXP (exhaustive search) but **not** in P
- Given a **Tetris** board + seq of pieces, can you **survive**?
  - In EXP but not **known** whether in P
- Given a computer **program** and **input**, does it **halt**?
  - Automated **infinite loop** checker
  - Uncomputable! ($\notin$ R)
  - **Cannot** solve in finite time for **all** programs, for **all** inputs

Halting problem

- Assume Halt(P, s) solves the **halting problem**:
  - Input two strings: a **program** P and an **input** s to P
  - Outputs TRUE ⇔ P(s) **halts**
- Now, consider the following **program**:

```
def Koan( X ):
  if Halt( X, X ):
    loop forever
Koan( Koan )
```

- Case 1: Koan(Koan) **halts**.
  - ⇒ Halt(Koan, Koan) = TRUE (by correctness of Halt)
  - ⇒ Koan(Koan) loops forever (by code of Koan)

Outline for today

- All-pairs shortest path
  - Johnson (reweighted Dijkstra): $O\left(|V|^2 \log|V| + |V||E|\right)$
- Tractability
  - Complexity classes: P, EXP, R
  - **Non-deterministic verification:** NP
  - **NP-hard and NP-complete**
- Semester review

NP

- **Certification** algorithm: instead of saying whether s $\in$ X,
  - Check a proposed **proof** t that s $\in$ X:
- C(s,t) is a **certifier** for problem X if for every s:
  - s $\in$ X $\Leftrightarrow$ $\exists$ **certificate** t such that C(s,t) = TRUE
- **NP** (nondeterministic polynomial): all problems X which have a **polynomial**-time **certifier**:
  - C(s,t) runs in polynomial time
  - Certificate t is of polynomial size: $|t| \in O(|s|^c)$
- **Nondeterministic** Turing machine:
  - Can make lucky **guesses** of t and **check** in polynomial time

Examples of NP

e.g., **Tetris** $\in$ NP:

- Given sequence of moves, can easily **check** if survive
- **Rules** are simple; **strategy** is hard

Most **games**: checkers, Minesweeper, Sudoku, etc.

Travelling salesman problem (TSP):

- **Shortest** path visiting **every** vertex in weighted graph
- **Decision** version: is min weight $\leq$ x?

Satisfiability (3-SAT):

- Boolean formula in **conjunctive normal form** (CNF):
  - $(x_1 \text{ or } x_2 \text{ or } x_3) \text{ and } (x_4 \text{ or } x_5 \text{ or } x_6) \text{ and } (x_7 \text{ or } x_8$
  - Each **clause** is an **OR** of up to 3 Boolean variables
  - CNF **formula** is an **AND** of all clauses

P vs NP

- $P \subseteq NP \subseteq EXP$
  - For any P problem, can **solve** $\Rightarrow$ can find **certificate** t
  - For any NP problem, can try **every string** t with $|t| < n$
- Million-dollar question (Clay prize): **P =? NP**
  - Is it easier to **check** a proof than **construct** one?
  - Can't "engineer luck"

Reductions

- **Convert** your problem into one with a known solution
  - **unweighted** shortest path → **weighted** shortest path:
    - set all edge weights to 1
  - **longest** path → **shortest** path (negate weights)
- Problem X **reduces** (poly, Cook) to Y, ($X \geq_p Y$) iff:
  - Can solve X using a polynomial number of **calls** to a solution of Y, plus polynomial additional work
  - Model of **computation** (subroutines)
  - X is "**at least** as hard as" Y

NP-hard and NP-complete

- X $\in$ **NP-hard** $\Leftrightarrow X \geq_p Y$ for all Y $\in$ NP
  - "**at least** as hard as NP"
- **NP-complete** = NP ⋂ NP-hard
  - all NP-complete problems are "the **same**" difficulty (mod poly)
  - If any **one** is in P, then they **all** are, and P = NP

NP complete

Examples of NP-complete

- 3-SAT, TSP, 0-1 knapsack (pseudo-poly)
- 3-partition: split n integers into triples of equal sum?
- 3-colouring of graphs (no adj nodes share colour)
- Find largest clique in a graph (fully connected subset)
- Shortest path in 3D avoiding **obstacles**
- Factoring integers is NP but **unknown** if NP-hard
- Chess (generalised to n x n) is EXP-complete
- **Protein** folding, urban **traffic flow** equilibrium, optimal **meshing** for FEM, max **social welfare** in Nash equilib, …
- Foundational **textbook**: Garey + Johnson, "Computers and Intractability" (1979)

Outline for today

- All-pairs shortest path
  - Johnson (reweighted Dijkstra): $O\left(|V|^2 \log|V| + |V||E|\right)$
- Tractability
  - Complexity classes: P, EXP, R
  - Non-deterministic verification: NP
  - NP-hard and NP-complete
- **Semester review**

Semester overview

- **Complexity**: V O l o ω, recurrence, induction (ch1-5)
- **Sorting** (ch4-8)
  - Comparison sorts: insert, merge, heap, quick
  - Linear sorts: counting, radix, bucket
- **Data Structures** (ch10-12, 18)
  - Hash tables, linked lists, BST*s, *B-trees
- **Divide and Conquer** (ch15-16)
  - Dynamic prog: rod cut, matrix-chain, LCS, opt BST
  - Greedy: act sel, list merge, Huffman, frac knapsack
- **Graph Algorithms** (ch22-25)

Lecture 1: ch1-3

- **Insertion** sort and its **analysis**
- Discrete **math** review
  - **Logic** and proofs
  - Monotonicity, limits, iterated functions
  - **Fibonacci** sequence and golden ratio
  - Factorials and **Stirling's** approximation
- **Asymptotic** notation: V , O, l , o, ω
  - **Proving** asymptotic bounds

Lecture 2: ch4-5

- **Divide and conquer** (ch4)
  - **Merge** sort and its **analysis**
  - Recursion trees + proof by **induction**
  - Maximum **subarray**
  - Matrix multiply vs **Strassen**'s method
  - **Master method** of solving recurrences
- **Probabilistic Analysis** (ch5)
  - **Hiring** problem and analysis
  - **Randomised** algorithms and PRNGs

Lecture 3: ch6-7

- **Heapsort** (ch6) :
  - Trees, binary **heaps**, **max-heap** property
  - **Heapify()** function on a node
  - **Heapsort**: build a max-heap, use it for sorting
  - **Priority queue** using max-heap: operations, complexity
- **Quicksort** (ch7) :
  - Regular quicksort with **fixed** (Lomuto) partitioning
  - **Randomised** pivot
  - Analysis of randomised pivot: **expected** time
- **Vegas**-style vs **Monte-Carlo** style probabilistic algo

Lecture 4: ch8,11

- **Linear**-time sorts (ch8) (**assumptions!**)
  - **Decision-tree** model, why comparison sorts are l (n lg n)
  - **Counting** sort: census + move: V (n+k), **stability**
  - **Radix** sort (with r -bit digits): V (d(n+k))
  - **Bucket** sort: V (n) **expected** time
- **Hash** tables (ch11):
  - Hash **function**, hash **collisions**, **chaining**
  - **Load factor** v = n/num_buckets
  - **Search** in V (1+v)
  - **Hashes**: div, mul, universal hashing
  - **Open addressing**: linear, quad, double-hash

Lecture 5-6: ch10,12,18

- **Linked lists** (ch10):
  - Singly/**doubly**-linked, **circular**
- **Stacks** and **queues** (ch10):
  - Operations, implementation with linked-lists
- **Trees** and Binary search trees (**BST**) (ch12):
  - Tree **traversals**: inorder, preorder, postorder
  - **Search**, **Min**/max and **successor**/pred
  - **Insert** and **delete**
  - **Randomised** BST
- **Skip lists**: implementation, complexity
- **B-Trees** (ch18)
  - **Search** / **insert** / **delete** in $O(t \log_t n)$

Lecture 8: ch15

- **Dynamic programming**
  - **Rod-cutting** problem
  - Proving optimal **substructure**
  - Recursive, top-down, **bottom-up** solutions
- **Fibonacci** sequence
- **Matrix-chain** multiplication
- Longest common **subsequence**
- Shortest unweighted **path**
- Optimal **binary search tree**

Lecture 9: ch16

- **Greedy** algorithms
  - Proving optimal substructure
  - Proving greedy property
- **Activity selection** problem
- **List merging** problem
- **Huffman** coding
- **Knapsack** problem: fractional and 0-1
- Optimal offline **caching**

Lecture 10: ch22

- **Graph** representation:
  - Edge list, adjacency list, adjacency matrix
- **Breadth-first** graph traversal
- **Depth-first** graph traversal
  - **Parenthesis** structure
  - Edge **classification**
  - Topological **sort**
  - Finding **strongly-connected** components

Lecture 11-12: ch23-25

- **Minimum spanning tree** (MST)
  - **Kruskal** (disjoint-set forest): $O(|E| \log |E|)$
  - **Prim** (priority queue): $O(|V| \log |V| + |E|)$
- **Single-source** shortest paths
  - **Bellman-Ford**: $O(|V||E|)$
  - Special case for **DAG** (no cycles)
  - **Dijkstra** (weights ≥ 0): $O(|V| \log |V| + |E|)$
- **All-pairs** shortest paths
  - **Dynamic** programming by path length: $O\left(|V|^4\right)$
  - **Exponential** speedup: $O\left(|V|^3 \log |V|\right)$
  - **Floyd-Warshall** (dyn prog by vertex subset):

Outline for today

- **All-pairs** shortest path
  - Johnson (reweighted Dijkstra): $O\left(|V|^2 \log|V| + |V||E|\right)$
- **Tractability**
  - Complexity **classes**: P, EXP, R
  - **Non-deterministic** verification: NP
  - NP-**hard** and NP-**complete**
- Semester **review**