COMP3821/COMP9801 Workshop Week 9

Convolutions and Flow Networks

Jeremy Le and Xueyi Chee

2025T3

Announcements

- Fill out your MyExperience survey for your experience of us!
- Problem Set 8 Submission due on Friday Week 9.
    - Submissions on Gradescope.
- Help sessions on Friday 11am - 1pm in K17 G05.

Guided Problem 1

$k = 3$

1 2 3

1 2 3 4

1 2 3

1 3 1

Guided Problem

It is the Year of the Snake, and you just so happen to have a collection of n coloured snake figurines. Each figurine is coloured one of c colours. Your friend wants to keep a set of k snake figurines from your collection. Two sets of snake figurines are considered the same if they contain the same number of figurines of each colour.

Describe and analyse a polynomial-time algorithm (in terms of c, k, n) that returns the number of different ways of choosing k snake figurines. Your input is an array $\text{SNAKE}[1..c]$ and an integer k, where $\text{SNAKE}[i]$ denotes the number of snake figurines of colour i. Your output is a non-negative integer.

Note. *You cannot assume that c or k are constants. You may assume that $k \leq n$.*

Convolutions

Definition of Convolution

Let A and B be two sequences. The **convolution** of A and B, denoted by $A \star B$, is the function (often viewed as a sequence) defined by

$$(A \star B)(t) = \sum_{i+j=t} A_i B_j.$$

Polynomial Coefficients and Convolution

Polynomial Coefficients

Let P_A and P_B be polynomials with coefficient sequences A and B respectively (padded to equal lengths if necessary).
Then we define their product polynomial

$$P_C = P_A P_B.$$

$(x^2 + x + 1)$
$(x + 1)$

The coefficient sequence C of P_C satisfies

$= x^3 + \boxed{2x^2} + 2x + 1$

$$C_t = (A \star B)(t) = \sum_{i+j=t} A_i B_j.$$

The Fast Fourier Transform (FFT)

FFT Result

In $\Theta(n \log n)$ time, we can:

- Compute the product $P_C(x) = P_A(x)P_B(x)$ of two polynomials P_A and P_B of degree $O(n)$.
- Find **all** the convolution values of two sequences A and B of (up to) n terms.

Watch the 3Blue1Brown video and everything will make sense.

Guided Problem 1 - Working

Input Snake[1 . . . c] , k

For colour j $\quad P_j(x) = x^{\text{Snake}[j]} + x^{s-1} + \ldots + 1$

$$= \sum_{i=0}^{\text{Snake}[j]} x^i$$

let $P(x) = \prod_{i=1}^{c} P_i(x)$ → find coeff x^k

return

number of ways of choosing one term from each polynomial such that the sum degrees equal k

Guided Problem 1 - Working

$a \times b \times c \times d$

$$p^{(i)}(x) = p^{(i-1)}(x) \times p_i(x)$$

Base Case $p^{(2)}(x) = p_1(x) \times p_2(x)$

$O(n \log n)$

Snake[1] + ... + Snake[c] = n

< multiplications each of running time $O(n \log n)$

$= O(cn \log n)$

Optimise by getting rid of any degree > k

$O(ck \log k)$

$O(ck)$ - via dynamic programming (I think)

Guided Problem 1 - Solution

Algorithm Outline

- Construct the polynomials for each colour j, as follows:

$$P_j(x) = \sum_{k=0}^{\textsc{Snake}[j]} x_k.$$

- Consider the following polynomial

$$P(x) = \prod_{j=1}^{c} P_j(x).$$

- Observe that the coefficient of x^k in $P(x)$ corresponds to the number of ways of choosing one term from polynomials P_j such that the chosen number of snake figurines sum to k.
- This gives us all the ways to choose k snake figurines.

Guided Problem 1 - Solution

Time Complexity

- First notice that

$$\textsc{Snake}[1] + \cdots + \textsc{Snake}[c] = n.$$

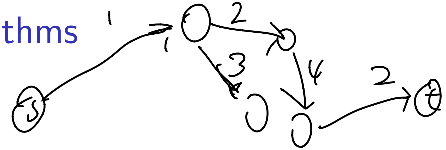
- To compute $P(x)$, this is just a series of pairwise products.
- We can compute $P^{(i)}(x) = P_i(x) \times P^{(i-1)}(x)$ in $O(n \log n)$ using the Fast Fourier Transform since $\deg P_i, \deg P^{(i-1)} \leq n$.
  - Base case: $P^{(2)}(x) = P_1(x) \times P_2(x)$ in $O(n \log n)$.
- Return $P^{(c)}(x)$, so there are c multiplications, gives $O(cn \log n)$.

Guided Problem 1 - Solution

Optimisation

- Optimise further by discarding all terms higher than degree k at every stage as these will not contribute towards the result.
- Then polynomial multiplications will have degree at most k, meaning each multiplication has running time $O(k \log k)$.
- Over c polynomial multiplications, gives $O(ck \log k)$ time.

Flow Algorithms



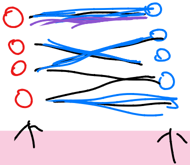
Suppose we have a flow network with maximum flow value $|F|$, $|V|$ vertices, and $|E|$ edges.

Flow Algorithms

- The Ford–Fulkerson algorithm has complexity $O(|E| \cdot |F|)$.
- The Edmonds–Karp maximum flow algorithm has complexity $O(|V| \cdot |E|^2)$, but it also satisfies the Ford–Fulkerson bound of $O(|E| \cdot |F|)$. Hence, its overall complexity is

$$O\big(\max\{|F| \cdot |E|,\ |V| \cdot |E|^2\}\big).$$

Bipartite Matching



Bipartite Graph

A graph $G = (V, E)$ is said to be **bipartite** if its vertices can be divided into two disjoint sets A and B such that every edge $e \in E$ has one endpoint in A and the other in B.

Matching

A **matching** in a graph $G = (V, E)$ is a subset $M \subseteq E$ such that each vertex of the graph is incident to at most one edge in M.

Bipartite Maximum Matching Problem

Given a bipartite graph G, find the size (i.e., the number of pairs matched) in a maximum matching.

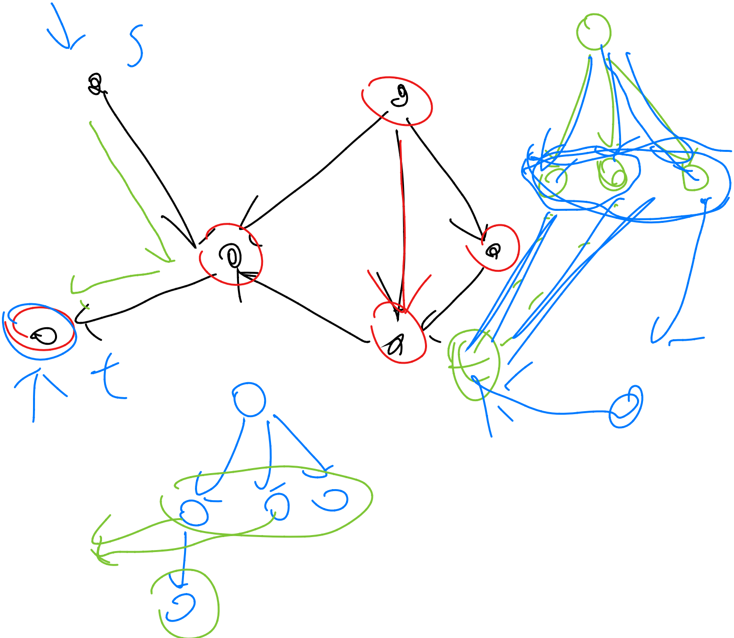
Guided Problem 2

Guided Problem

Let $G = (V, E)$ be a *d*irected and acyclic graph, where $|V| = n$ and $|E| = m$. Let $\mathscr{P} = \{P_1, \ldots, P_k\}$ be a collection of disjoint paths in G. We say that $\mathscr{P}$ is a path covering if every vertex $v \in V$ in G is covered in *e*xactly one path P_i. In other words, a path covering is a collection of disjoint paths that cover all vertices in G.

Describe and analyse a polynomial-time algorithm that returns the minimum number of paths such that each vertex in V lies in *e*xactly one path P_i.

Guided Problem 2 - Working

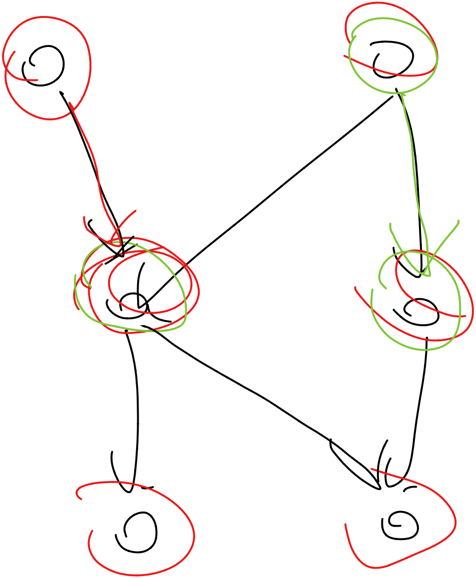


Guided Problem 2 - Working

n

n k

n - k

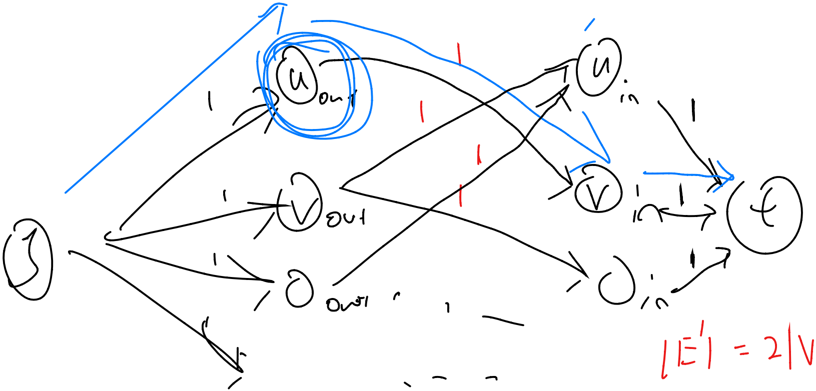


1) each node only pick at most 1 out edge

2) ∴ pick at most 1 in edge

Guided Problem 2 - Working

Out $u \to v$



$|E'| = 2|V| + |E|$

$|f| = |V|$

$O(\quad)$

$O(|E||f|)$

f

$O((|V|+|E|)|V|) = O(m(m+n))\ \underline{n-f}$

Guided Problem 2 - Solution Outline

Algorithm and Time Complexity

- Let $G' = (V', E')$ be the graph constructed as follows:
  - Construct two vertices s and t.
  - For each vertex $v \in V$, construct vertices v^{in} and v^{out}.
  - For every edge $(u, v) \in E$, construct edge $(u^{\text{out}}, v^{\text{in}})$ in G'.
  - For each vertex v^{out}, construct edge (s, v^{out}) with capacity 1.
  - For each vertex v^{in}, construct edge (v^{in}, t) with capacity 1.
- Run Ford-Fulkerson to find maximum bipartite matching in $O(|E'||V'|)$. Then note,

$$|V'| = 2|V| \quad \text{and} \quad |E'| = |E| + 2|V|.$$

- Therefore time complexity is $O((m + n)n)$.

Guided Problem 2 - Solution Outline

Proof - Forward Direction

Claim.

G has a path covering of size k $\iff$ G' has a matching of size $n-k$.

$(\implies)$ Suppose G has path covering $\mathscr{P}$ of size k. In each path, there is exactly one vertex with in-degree 0. Therefore, $\mathscr{P}$ has exactly $n-k$ edges. Now consider the subset of edges,

$$M = \{(u^{\text{out}}, v^{\text{in}}) \in E' : (u,v) \in \mathscr{P}\}.$$

Every vertex of G' corresponds to at most one incoming and outing edge in $\mathscr{P}$. Every vertex is incident to at most one edge in M (matching). Each element of M corresponds to unique element of $\mathscr{P}$. So M is a matching of size $n-k$. $\square$

Guided Problem 2 - Solution Outline

Proof - Reverse Direction

Claim.

G has a path covering of size $k \iff G'$ has a matching of size $n-k$.

($\Longleftarrow$) Consider a matching M of size $n-k$. Each vertex u such that u^{in} is not an endpoint of some $m \in M$ must be the start of a distinct path $P \in \mathscr{P}$. Since $|M| = n-k$, we must have $|\mathscr{P}| = k$.

Since M is a matching, each edge $m \in M$ such that $m = (u^{\text{out}}, v^{\text{in}})$ can be sequenced after u^{in} for exactly one $P \in \mathscr{P}$. We can terminate each path by exactly one of k vertices v such that v^{out} is not an endpoint of some $m \in M$. Each vertex is included in $\mathscr{P}$, so forms a valid path covering of size k. $\square$

Dilworth and Mirsky for Exercise 4

Dilworth's Theorem

Let $(P, \preceq)$ be a finite partially ordered set. Then the maximum size antichain of P equals to the minimum number of chains that partition P.

Mirsky's Theorem

Let $(P, \preceq)$ be a finite partially ordered set. Then the maximum size chain of P equals the minimum number of antichains that parition P.

Mirsky's theorem is the dual of Dilworth's theorem.