Surname:

First Name:

Matriculation No.:

Sample Solution

Karlsruhe Institute of Technology
Institute for Theoretical Informatics

Prof. Dr. Marvin Künnemann 18.09.2026

Exam Algorithms II

Problem 1.	Miscellaneous Tasks	13 Points
Problem 2.	Max-Flow: Dynamic Max Flow	10 Points
Problem 3.	Shortest Paths: Alien Abduction	8 Points
Problem 4.	Stringology: Patterns and Prefixes	9 Points
Problem 5.	Geometric: Discrete Fishing	10 Points
Problem 6.	Approximation: Tourist Taxi Toll	10 Points

Please note:

- The only allowed aid is **one** A4 sheet with your **handwritten** notes.
- Write your answers in blue or black ink only, using an indelible pen.
- **Write** your **Matriculation No.** on **all** sheets of the exam and additional pages.
- The duration of the exam is **120 minutes**.
- You may write on the backside of the sheets. If you do so, please reference the exercise that you answer.
- The exam contains **17 pages**.

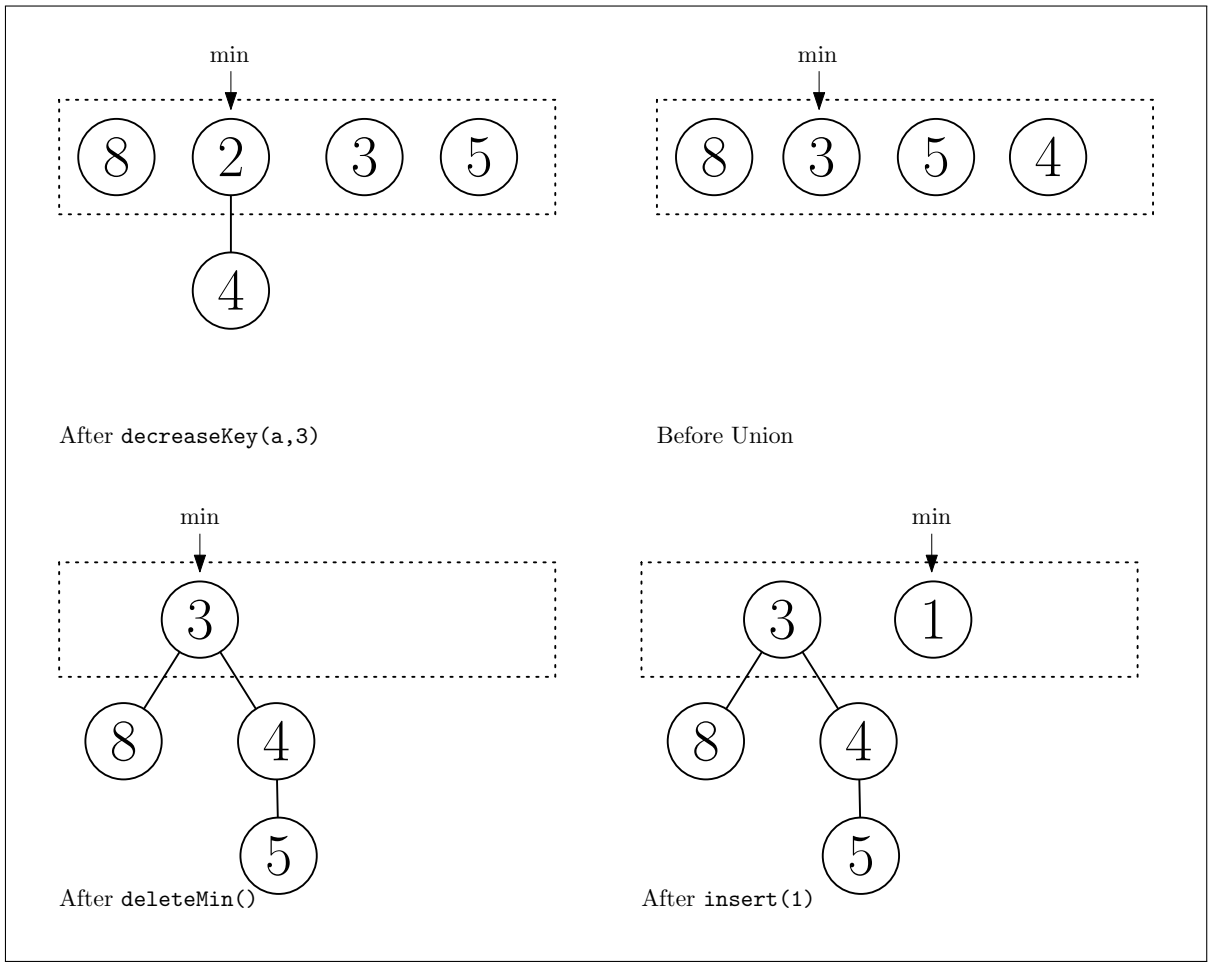
Problem 1. Miscellaneous Tasks **(_ /13 P.)**

a. Given a Fibonacci Heap in the state depicted below, perform the following operations on it. The ✗ denotes a node as marked and the min-ptr the current minimum. Draw the resulting state (including any marks and the min-ptr) after every operation **and right before union-by-rank**. **(_ /4 P.)**

Hint: You should draw 4 states in total.

1. `decreaseKey(a,3)`
2. `deleteMin()`
3. `insert(1)`

Solution



b. Consider algorithms with the following running times for input size n and parameter $k \in \mathbb{N}$. **(_ /3 P.)**
State whether they are fixed-parameter tractable times and justify your answer.

1. $(k!) \cdot n^{1+\log k}$
2. $\frac{(n-k)^k}{2^k}$
3. $(k \cdot n \log n)^3$

Solution

1. $(k!) \cdot n^{1+\log k}$ is not FPT as the exponent of n depends on $\log k$.
2. $\frac{(n-k)^k}{2^k} = \Theta\left(\left(\frac{n}{2}\right)^k\right)$ is not FPT as the exponent of n depends on k.
3. $(k \cdot n \log n)^3$ is FPT as $f(k) = k^3$ and $p(n) = (n \log n)^3$.

c. Consider approximation algorithms with the following running times for input size n and approximation factor $(1+\varepsilon)$ for some minimization problems. State whether they are an FPTAS, a PTAS but not an FPTAS, or neither. Justify your answer. **(_ /2 P.)**

1. $\frac{1}{\varepsilon} \cdot n^{\frac{1}{\varepsilon}^2}$
2. $n^2 \cdot \left(\frac{1}{\varepsilon}\right)^{\sqrt{n}}$

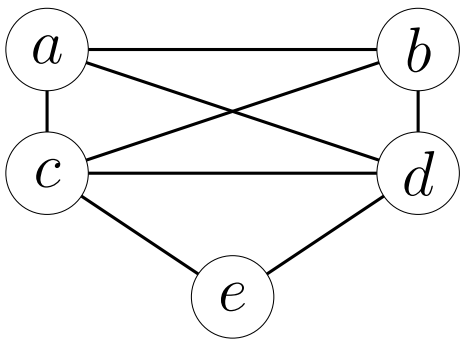
Solution

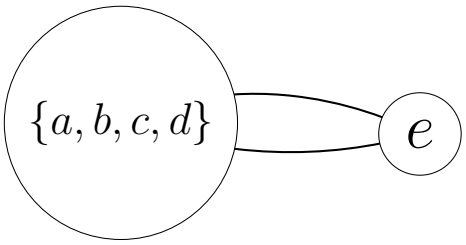
1. $\frac{1}{\varepsilon} \cdot n^{\frac{1}{\varepsilon}^2}$, is a PTAS but not a FPTAS, as it is polynomial in n, but not in $\frac{1}{\varepsilon}$.
2. $n^2 \cdot \left(\frac{1}{\varepsilon}\right)^{\sqrt{n}}$ is neither, as the $\left(\frac{1}{\varepsilon}^{\sqrt{n}}\right)$-term is already not polynomial in n.

Sample Solution

d. Consider the following unweighted and undirected graph. **(_ /2 P.)**
Give a contraction order for the edges, such that Karger's algorithm yields a minimum cut of the graph (when contracting edges in this order). Draw the resulting multigraph.
Tip: You can denote the edges by their vertices in the starting graph even after contracting.

Solution





contraction order = ab, bc, bd

e. Assume you are given a large list S of n primitive (comparable) elements that does not fit into main memory of size M. Let B be the block size. Show that you can count the number of distinct elements in S with $\mathcal{O}\left(\frac{n}{B}\left(1+\log_{M/B}\frac{n}{M}\right)\right)$ I/O-accesses. **(_ /2 P.)**

Solution

We use an external sorting algorithm, that uses $\mathcal{O}\left(\frac{n}{B}\left(1+\log_{M/B}\frac{n}{M}\right)\right)$ I/O-accesses as discussed in the lecture. With $\mathcal{O}(n/B)$ I/O-accesses, we scan the sorted list blockwise, counting the number of distinct elements by counting the number of element changes (also between blocks).

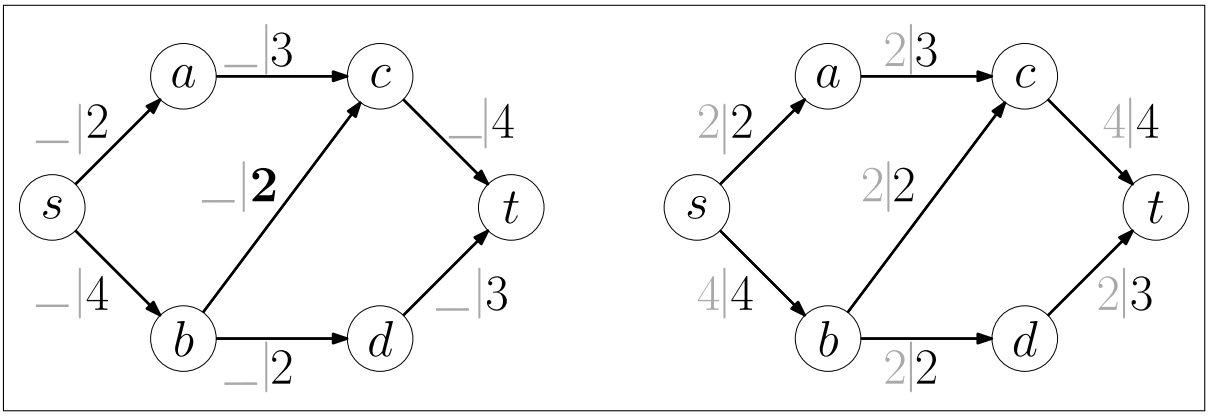
Sample Solution

Problem 2. Max-Flow: Dynamic Max Flow **(_ /10 P.)**

In the following we are working with a flow network $(G = (V,E), c, s, t)$ with integer capacities $c : E \to \mathbb{N}_0$. As usual, we denote the source and sink by $s, t \in V$.

a. Given the following network with a maximum flow f with edges labelled by $f(e)|c(e)$, compute the new maximum flow after increasing the edge capacity as shown below. **(_ /1 P.)**

Solution



In the following tasks, you are given the network and an integer maximum flow f in G.

b. Design an algorithm that, given an edge $e \in E$, **increases** its capacity by 1 and updates f such that it is still a maximum flow. Your algorithm should run in time $\mathcal{O}(n+m)$. Prove the correctness of your algorithm and analyze its runtime. **(_ /3 P.)**

Solution

First, we update the edge capacity of e. f is still valid, as we increased the capacity. A single DFS/BFS run suffices to determine an augmenting path if it exists, and it is left to update f accordingly.
As finding the augmenting path dominates the runtime, we obtain a $\mathcal{O}(n+m)$ algorithm.
As we increase the edge-capacity by exactly 1 and f previously was an integer max-flow, it can improve by at most one. Thus, it suffices find one augmenting path, if it exists. Otherwise, f is already a maximum flow.

c. Design an algorithm that, given an edge $e \in E$ with $c(e) \geq 1$, **decreases** its capacity by 1 and updates f such that it is still a maximum flow. Your algorithm should run in time $\mathcal{O}(n+m)$. Prove the correctness of your algorithm and analyze its runtime. **(_ /4 P.)**

Note: If it helps, you may assume that there is no cycle with all edges having positive flow under f.

Solution

If e is not saturated, then decreasing its capacity maintains feasibility of f and thus it is a maximum flow.
Otherwise, it f is no longer feasible. We determine a path through e in linear time $n+m$ that pushes one unit of flow (i.e. search s-u and v-t-paths for $(u,v) = e$ using DFS) and decrease the flow of f along this path. This make f feasible again.
Finally, we check whether there exists an augmenting path in $\mathcal{O}(n+m)$, updating f accordingly. The algorithm runs in total time $\mathcal{O}(n+m)$ using several DFS.
If e was not saturated, then correctness is clear. Otherwise, by capacity bound of $c(e) \geq 1$ there exists a path that pushes at least one unit of flow over e. Therefore, we can reduce the flow on this path to make f feasible again. As f was a maximum flow in the original graph, the new maximum flow not increase over it. Thus, it is sufficient to find a single augmenting path. Otherwise, the modified f is a maximum flow.

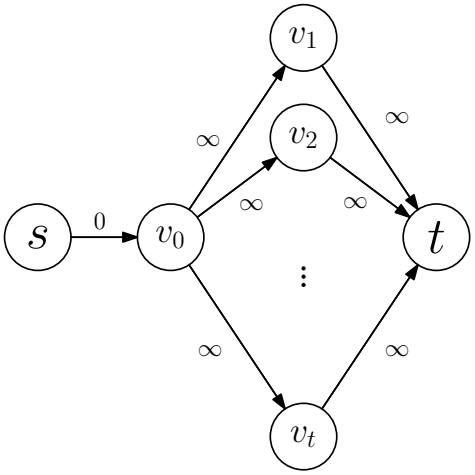
d. Show that there exists an infinite sequence of networks $G_1, G_2, \ldots$ where network G_x contains some edge e such that increasing the capacity of e by x results in x distinct augmenting paths in the residual graph. It suffices to describe how to construct G_x for any given x. You do not need to prove that G_x satisfies this property. **(_ /2 P.)**

Solution

We construct G_x in the following manner. Let $V = \{s,t\} \cup \{v_0, v_1 \ldots, v_x\}$ and construct the following edges:

$$E = \{(s,v_0),(v_0,t)\} \cup \{(v_0,v_i) \mid i \in [x]\} \cup \{(v_i,t) \mid i \in [t]\}.$$

Let $c(e') = \infty$ for all $e' \in E$ except $c((s,v_0)) = 0$. Then increasing $c((s,v_0))$ by x yields exactly x many augmenting paths $P_i = s, v_0, v_i, t$ for all $i \in [x]$ that clearly are all distinct.



Sample Solution

Problem 3. Shortest Paths: Alien Abduction **(_ /8 P.)**

Two scientists have been tracking alien abductions throughout Baden-Württemberg. They managed to precisely calculate the probability of **not** being abducted by aliens when traveling from one city to another.

To this end, they have constructed a directed graph $G = (V, E, p)$, where the vertices V represent cities and the edges E represent direct connections between cities. Each edge $(u, v) \in E$ is labelled by the probability $p(u, v)$ of not being abducted when traveling from city u to city v. As the aliens are very peculiar about numerical values, the probabilities only take values $p(u, v) = \frac{1}{2^k}$ for some $k \in \mathbb{N}_0$.

For a starting city and a destination city, the probability of not being abducted is given by the product of the probabilities along the edges of the path taken. We denote the probability of not being abducted on a path $P = s, e_1, v_1, \ldots, v_{l-1}, e_l, t$ as its *safety*:

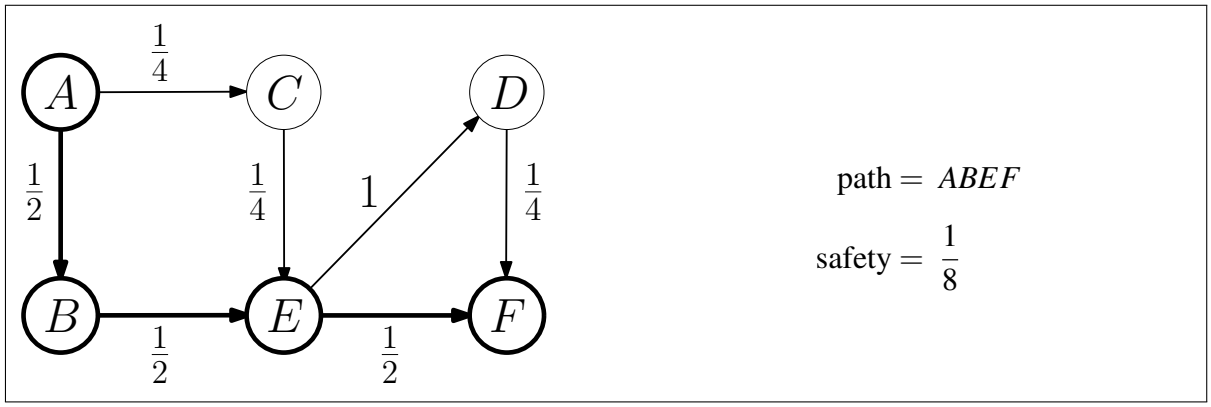
$$\text{safety}(P) = \prod_{i=1}^{l} p(e_i).$$

Finally, for two cities s, t we denote by $\text{safety}(s, t)$ the maximum safety over all s-t-paths in G. If there is no path from s to t we define $\text{safety}(s, t) = 0$. Finally, for a city $s \in V$ we define $\text{safety}(s, s) = 1$.

Note: You can use the previous subtasks to solve later problems, even if you did not answer them.

a. Consider the labelled graph given below. Compute the safest path from city A to city F and state $\text{safety}(A, F)$. **(_ /1 P.)**

Solution



path = $ABEF$

safety = $\frac{1}{8}$

Sample Solution

b. Given $G = (V, E, p)$ where $p : E \to \{1, \frac{1}{2}, \frac{1}{4}, \frac{1}{8} \dots\}$, design an algorithm that, given a starting city $s \in V$, computes safety(s,t) for all $t \in V \setminus \{s\}$ in time $\mathcal{O}(m + n \log n)$. Prove the correctness and analyze the runtime of your algorithm. **(_ /3 P.)**

Solution

Replace every edge weight $p(u,v)$ by $c(u,v) = -\log(p(u,v))$. Then, the safety of a path P is given by

$$\prod_{(u,v)\in P} p(u,v) = \prod_{(u,v)\in P} e^{-c(u,v)} = e^{-\sum_{(u,v)\in P} c(u,v)}.$$

Therefore, minimizing $-\sum_{(u,v)\in P} c(u,v)$ yields the safest path: the safest path from s to v is given by the path

$$P^* = \arg \min_{P \in \mathcal{P}(s,v)} \sum_{(u,v)\in P} c(u,v),$$

where $\mathcal{P}(s,v)$ is the set of all paths from s to v. As $c(u,v) \geq 0$, we can use Dijkstra's algorithm (using Fibonacci heaps) to compute the safest path from s to every other vertex $v \in V \setminus \{s\}$ in time $\mathcal{O}(m + n \log n)$. Correctness follows from Dijkstra's and our previous argument.

c. Due to unforeseen circumstances, the scientists are split up and find themselves in two different cities s_1 and s_2. They want to meet in a city t while trying to ensure the most dangerous of the two journeys is as safe as possible. **(_ /3 P.)**
Given $G = (V, E, p)$ where $p : E \to \{1, \frac{1}{2}, \frac{1}{4}, \frac{1}{8} \dots\}$ and starting cities $s_1, s_2 \in V$, design an algorithm that computes the safest meeting point $t \in V$ such that $\min(\text{safety}(s_1, t), \text{safety}(s_2, t))$ is maximized. Your algorithm should run in $\mathcal{O}(m + n \log n)$ time. Prove the correctness and analyze the runtime of your algorithm.

Solution

We use the algorithm describe in the previous subtask with starting cities s_1 and s_2 each. By reflexivity of safety we can create two arrays A_1, A_2 of length n containing distances from s_1 and s_2 to any $t \in V$ respectively. Computing the minimizer over all $t \in V$ can therefore be done in linear time, by checking each the corresponding array entries.
The running time amounts to two runs of subtask b), storing the result in A_1, A_2 and a single iteration over all $t \in V$, totaling a runtime of $m + n \log$.
Correctness follows from subtask b) and a full enumeration of all possible $t \in V$.

d. Can there exist a safest s-t-path (not necessarily simple) that contains a cycle? Briefly argue your answer. **(_ /1 P.)**

Solution

Yes. A safest s-t-path P can only contain a cycle C, if $\text{safety}(C) = 1$, otherwise P is strictly better when not taking C.

Sample Solution

Problem 4. Stringology: Patterns and Prefixes **(_ /9 P.)**

We introduce a generalization of the String Matching Problem, in which the pattern P is allowed to have the wildcard-symbol $\star$ which matches any number of characters (including the empty string).
A pattern $P \in (\Sigma \cup \{\star\})^*$ with wildcards $\star$ matches a string $X \in \Sigma^*$ if by replacing each occurrence of a wildcard $\star$ in P by some (individual) string over Σ, we can obtain a substring of X.

a. Consider the following string $X =$ `ABERACADABERA`. Finish the following table by indicating whether the pattern matches X: **(_ /1 P.)**

Solution

Pattern	matches X?
`D⋆C`	×
`BE⋆C⋆BE`	✓
`A⋆B⋆C⋆D⋆E`	✓

b. Describe an algorithm that, given a string $X \in \Sigma^*$ of length n and pattern $P \in (\Sigma \cup \{\star\})^*$ of length m that contains q wildcards $\star$, decides whether P matches X. **(_ /4 P.)**
Your algorithm should run in $\mathcal{O}((q+1)n+m)$ time. Prove the correctness and analyze the running time.

Solution

We begin by splitting the pattern P into $q+1$ parts P_i at the wildcard $\star$. Let $m_i = |P_i|$. We successively run KMP on patterns P_i, starting at index $s_{i-1} + m_{i-1}$, where S_{i-1} is the first matched index of the previous pattern S_{i-1}. For P_0 we start at index 0. If we find a match for final pattern P_q, we return yes, otherwise there is some pattern P_i that was not matched, and we return no.
The algorithm runs in time

$$\mathcal{O}\left(\sum_{i=1}^{q+1}(n-(s_{i-1}+m_{i-1}))+m_i\right) \leq \mathcal{O}\left(\sum_{i=1}^{q+1} n + \sum_{i=1}^{q+1} m_i\right)$$
$$\leq \mathcal{O}((q+1)n+m)$$

when every pattern matches. Otherwise, we return early.
Correctness follows from the fact that for P_i there has to be at least one occurrence of P_j in X for every $j < i$. As our algorithm chooses the first occurrence for a pattern P_i, we continue searching in the maximal length suffix, thus we can not miss a match.

Sample Solution

c. You are given a dictionary of strings $D \subseteq \Sigma^*$ stored in a trie. We want to compute for a given string $w \in \Sigma^*$ the number of strings $d \in D$ such that w is a prefix of d or d is a prefix of w. **(_ /4 P.)**

You are allowed to save additional information to the nodes of the trie, if your modifications do not change the asymptotic running times of its operations.

Describe an algorithm (and modifications to the data structure) that answers such a query in time $\mathcal{O}(|w|)$. Prove the correctness and analyze the running time of your algorithm.

Solution

We modify tries in the following way: for every node, store the number of terminals in its subtree (not counting the node itself). This information can be maintained by updating this value when traversing the trie during the operations, thus only adding constant overhead per visited node.

We query the modified trie using w as usual. Let w_{prefix} be the number of terminals on the path.

If we successfully traverse all characters of w, let w_{suffix} be the number stored in the final visited node. If the traversal stops early because a character is not found, w cannot be a prefix of any word in D, so we set $w_{\text{suffix}} = 0$. The algorithm then returns $w_{\text{prefix}} + w_{\text{suffix}}$.

The running time is the same as a simple lookup operation with constant overhead per node, thus it runs in $\mathcal{O}(|w|)$.

As every terminal on our lookup path is a word $d \in D$ such that d is a prefix of w and the stored information in a node gives us the exact number of suffixes $d \in D$, the returned number is indeed the total number strings $d \in D$ such that w is a prefix of d or d is a prefix of w.

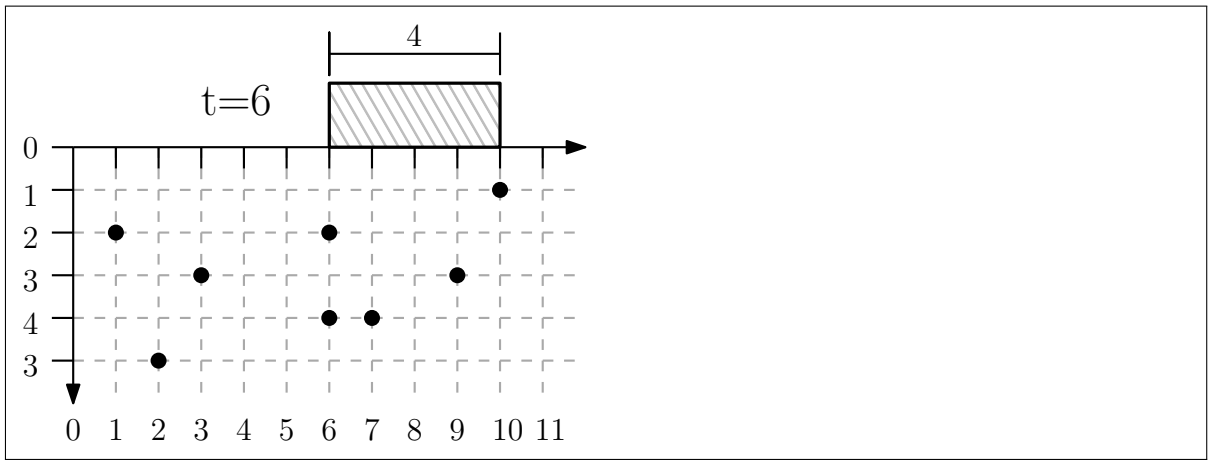
Sample Solution

Problem 5. Geometric: Discrete Fishing **(_ /10 P.)**

We want to help a crew of fishers secure their dinner in the Integer-Ocean spanning the coordinates $\mathbb{Z}_{\geq 0} \times \mathbb{Z}_{\geq 0}$. The ship sails along the surface at $y = 0$, has a length of $L \in \mathbb{N}$, and its position is defined by the coordinate $t \in \mathbb{Z}_{\geq 0}$ of its leftmost side. Thus at position t, the ship *covers* the $L+1$ integer x-coordinates $\{t, t+1, \ldots, t+L\}$.
There are m fish in the sea, located at (integer) coordinates $F = \{(x_1, y_1), \ldots, (x_m, y_m)\}$. The crew is equipped with highly advanced, infinite-length fishing rods that drop straight down parallel to the y-axis. If a rod is *deployed* at a surface coordinate x_r currently covered by the ship, it will catch all fish located at (x_r, y) for $y \in \mathbb{Z}_{\geq 0}$. As the infinite-length fishing line is very delicate, they can only throw out their rods if the ship is at rest.

a. Assuming the crew can deploy an unlimited number of fishing-rods, what is the optimal position t for the ship such that the maximum number of fish can be caught? **(_ /1 P.)**

Solution



b. Assume the crew can deploy an unlimited number of fishing-rods. Design an algorithm running in time $\mathcal{O}(m \log m)$ that, given ship length L and fish coordinates F, determines an optimal discrete position t such that the largest number of fish can be caught at this position. Prove the correctness and analyze the runtime. **(_ /4 P.)**

Solution

Because the fishing rods have infinite length, the depth (y-coordinate) of the fish is irrelevant. We can project all fish onto the 1D surface, representing them solely by their x-coordinates $\{x_1, \ldots, x_m\}$.
A fish at coordinate x_i is covered by the ship if and only if $t \leq x_i \leq t + L$. Thus, it can be caught if the ship's position t falls in the integer interval $[x_i - L, x_i]$. Finding the best ship position is equivalent to finding the integer coordinate with the maximum interval overlap.

1. For each fish at x_i, create two events: an `Enter` event at coordinate $x_i - L$, and an `Exit` event at coordinate $x_i + 1$ (since the interval is inclusive, the overlap drops after x_i).
2. Sort all $2m$ events in ascending order by their coordinate. If an `Enter` and an `Exit` event share the same coordinate, process the `Exit` event first.
3. Sweep from left to right through the sorted events, maintaining a counter (initialized to 0).
4. Increment the counter upon an `Enter` event, and decrement upon an `Exit` event. Track the maximum counter value and the coordinate t where it occurs.

Correctness & Runtime: Correctness follows from the 1D sweepline processing boundaries exactly where overlap counts change. Creating the events takes $\mathcal{O}(m)$. Sorting takes $\mathcal{O}(m \log m)$ time. The single sweep pass takes $\mathcal{O}(m)$ obtaining a total runtime of $\mathcal{O}(m \log m)$.

Sample Solution

c. It turns out the crew members are very peculiar: they only sit at fixed integer offsets $P \subseteq \{0,\dots,L\}$ relative to the left side of the ship. If the ship is at position t, the crew member at offset $p \in P$ can deploy their fishing-rod only at position $t+p$. **(_ /5 P.)**

Let $X = \max_{(x_i,y_i)\in F} x_i$ be the maximum x-coordinate of any fish.

Design an algorithm that, given ship length L, fish coordinates F and crew seats P, computes a ship position $t \in \mathbb{Z}_{\geq 0}$ that maximizes the total number of fish caught by the crew. Your algorithm should run in time $\mathcal{O}(m+(L+X)\log(L+X))$.

Prove the correctness and analyze the runtime.

Solution

We model the projected fish positions and the crew seating as polynomials. Multiple fish can share the same x-coordinate, so we must account for multiplicities.

1. Create a polynomial for the fish: $A(x) = \sum c_k x^k$, where c_k is the total number of fish located at x-coordinate k. The degree of this polynomial is d.
2. Create a polynomial for the crew seats, but reversed: $B(x) = \sum_{p\in P} x^{L-p}$. The degree is L.
3. Multiply the two polynomials $C(x) = A(x) \cdot B(x)$ using the Fast Fourier Transform (FFT).

Correctness: The coefficient of x^k in the product $C(x)$ is the sum of products of coefficients from A and B where the exponents sum to k. Thus, a fish at x_i and a seat at $L-p$ contribute to the term $k = x_i + L - p$. Rearranging this yields $x_i - p = k - L$.
If we set the ship's position to $t = k-L$, this matches the catching condition $x_i = t+p$. Thus, the coefficient of x^{t+L} in $C(x)$ represents exactly the number of fish caught when the ship is at position t. We simply find the maximum coefficient in $C(x)$ for exponents $k \geq L$, and return $t = k-L$.
Runtime: Constructing the polynomials takes $\mathcal{O}(m+L)$ time. The multiplication using FFT takes $\mathcal{O}((L+d)\log(L+d))$ time. Scanning the resulting coefficients takes $\mathcal{O}(L+d)$ yielding an overall runtime in $\tilde{\mathcal{O}}(m+(L+d)\log(L+d))$.

Sample Solution

Problem 6. Approximation: Tourist Taxi Toll **(_ /10 P.)**

You are a taxi driver who offers sightseeing tours for tourists. Instead of taking the shortest path, you want to find a route that visits every landmark exactly once and returns to the starting vertex while **maximizing** the total fare of the tour (luckily the tourists are too busy being amazed by the sights to care).

Let $G = (V,E)$ be a complete undirected graph with edge weights $w : E \to \mathbb{N}_0$, which models the landmarks as vertices, the roads as edges and the fare between two landmarks as the edge weight. You want to find the *maximum fare tour* $T \subseteq E$ that visits every vertex once and returns to the starting vertex, while maximizing the total edge weight $w(T) = \sum_{e \in T} w(e)$.

a. Given the graph below, state the total fare of the maximum fare tour. **(_ /1 P.)**

b. The local authorities are getting suspicious of your exorbitant fares. You want to prove to them that it is in fact (under plausible assumptions) too hard to maximize the fare, so their suspicion is unfounded. **(_ /4 P.)**

Show that there exists no polynomial time $\alpha(n)$-approximation algorithm A with a ratio $\alpha(n) > \frac{n-1}{n}$ for finding the maximum fare tour, unless P = NP.

Hint: You can use that Hamiltonian Cycle is NP*-complete: Given an undirected graph, the problem asks whether there is a tour T that visits every vertex once.*

Solution

Let $G = (V,E)$ be an instance for Hamiltonian Cycle. We construct an instance G' for maximum fare tour.

$G' = (V, V \times V)$ where

$$w(\{u,v\}) = \begin{cases} 1 & \text{if } \{u,v\} \in E \\ 0 & \text{otherwise} \end{cases}.$$

A Hamiltonian Cycle T in G thus is a maximum fare tour T' of weight n in G'. Conversely, if there is no Hamiltonian Cycle in G then the best tour in G' is of weight at most $n-1$.

Assume there is a polynomial-time approximation algorithm A with a ratio $\alpha(n) > \frac{n-1}{n}$. If G has a Hamiltonian Cycle (OPT $= n$), the algorithm guarantees $A(G') \geq \alpha(n) \cdot n > \frac{n-1}{n} \cdot n = n-1$. Since edge weights are integers, $A(G') = n$. If G has no Hamiltonian Cycle (OPT $\leq n-1$), the algorithm can return at most $n-1$. Thus running A can distinguish whether there exists a Hamiltonian Cycle in G.

As this reduction runs in time linear in the size of G, such an algorithm would solve Hamiltonian Cycle in polynomial time, meaning P = NP.

The authorities believed your proof and dismissed their suspicions.
Unfazed and undisturbed you proceed with your daily hustle using the following greedy algorithm:

Algorithm $\mathscr{A}$

Input: Complete undirected graph $G = (V, E)$ with non-negative edge weights w
Output: approximate maximum fare tour $T \subseteq E$

$T \leftarrow \emptyset$
$U \leftarrow V$ ▷ Set of unvisited vertices
Pick an arbitrary starting vertex $s \in U$ and let $v := s$
$U \leftarrow U \setminus \{v\}$
while $U \neq \emptyset$ **do**
    Pick a vertex $v' \in U$ that maximizes $w(\{v, v'\})$
    $T \leftarrow T \cup \{\{v, v'\}\}$
    $U \leftarrow U \setminus \{v'\}$
    $v \leftarrow v'$
$T \leftarrow T \cup \{\{v, s\}\}$
return T

Prove that algorithm $\mathscr{A}$ reaches a $\frac{1}{2}$-approximation ratio. To this end, order the vertices $\{v_1, \ldots, v_n\}$ as visited by the greedy tour T. We now assign the following sets of edges to each vertex v_i for $i \in \{1, \ldots, n-1\}$ based on this ordering:

$$E_i = \{\{v_i, v_j\} \mid j > i\}.$$

Let OPT be an optimal tour in the following.

c. Show that $|\text{OPT} \cap E_i| \leq 2$ for every $i \in \{1, \ldots, n-1\}$. **(_ /1 P.)**

Solution

As OPT visits every vertex only once, every v_i can appear at most 2 times in total over all edges in OPT. As E_i contains only edges with vertex v_i it thus follows that $|\text{OPT} \cap E_i| \leq 2$ for all $i \in [n]$.

d. Show for $i \in \{1, \ldots, n-1\}$ that $\max_{e \in E_i} w(e) \leq w(\{v_i, v_{i+1}\})$ for the given ordering. **(_ /1 P.)**

Solution

Assume we consider v_i in the greedy algorithm. Then $\mathscr{A}$ adds vertex v_{i+1} to the tour if and only if $w(v_i, v_{i+1})$ is maximal over all other unvisited vertices. At this point the set of unvisited vertices is exactly $U = \{v_{i+1}, \ldots v_n\}$. Thus $w(v_i, v_{i+1}) \geq \max\{w(v_i, v_j) \mid i < j\} = \max_{e \in E_i} w(e)$ proving the claim.

e. Using the previous subtasks, show that $\mathscr{A}$ has an approximation ratio of $\frac{1}{2}$. **(_ /3 P.)**

Solution

The weight of an optimal tour is given by

$$\begin{aligned}
&\sum_{e\in \mathrm{OPT}} w(e)\\
&\leq \sum_{i=1}^{n-1} \sum_{e_i\in E_i\cap \mathrm{OPT}} w(e_i)\\
&\leq \sum_{i=1}^{n-1} |E_i\cap \mathrm{OPT}| w(v_i,v_{i+1})\\
&\leq 2\sum_{i=1}^{n-1} w(v_i,v_{i+1})\\
&\leq 2\sum_{e\in T} w(e)
\end{aligned}$$

Using this, the approximation ratio can be obtained by

$$\frac{w(T)}{w(\mathrm{OPT})} \geq \frac{1}{2}.$$