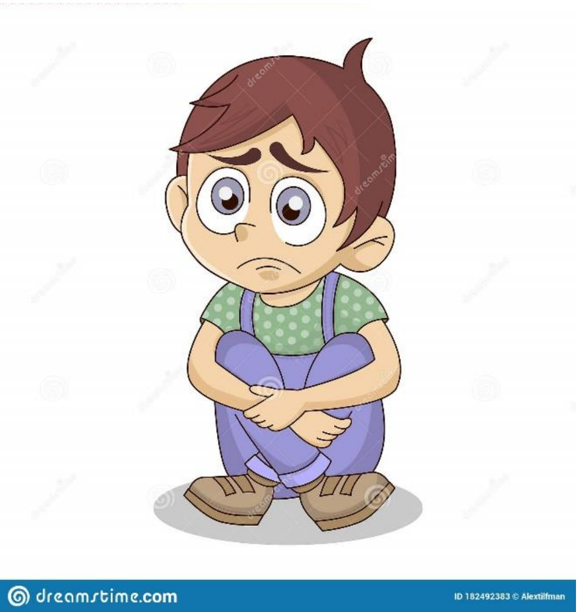
CSE 421

Final Review

Yin Tat Lee



Please fill in the evaluation.
Response rate is 33% only.
https://uw.iasystem.org/survey/253736

Final Exam

Format:

~20% True / False

~20% Fill in the blank

~60% 5 questions (1 question no proof)

Time: 2:30-4:20 (Mon, Mar 14) (110min)

Location: CSE2 G20 (same room)

Open book, open note.

Coverage: Lecture 1 – 24 (everything up to NP completeness)

Topics: Graph, Greedy, Divide and Conquer, Dynamic Programming, Maxflow, NP completeness.

Tips:

Please come up some algorithms for all questions (even if it is slower or may not work.)

Knowing how to greedy in real life is important.

Question

Consider the following decision problems:

Problem A

Input: graph G with vertex s, t, capacity c and integer k.

Output if the maxflow value from s to t is at least k.

Problem B

Input: graph G with vertex s, t, capacity c and integer k.

Output if the maxflow value from s to t is at most k.

Show that problem A and B are in NP.

(Furthermore, we require the certifier takes linear time.)

Answer for problem A

Algorithm:

// G, s, t, c, k are the input, f is the certificate

Function C(G, s, t, c, k, f)

check if f is a s-t flow on G with flow value $\geq k$

If **true**, return **yes**, else return **no**.

// "Certifier returns no" does not mean the maxflow $< k$.

// It only means the certificate is not valid

Runtime: $O(m)$ by going through all edges of G.

Proof:

If maxflow value $\geq k$, then we have a flow with value $\geq k$. Hence, we have the certificate f.

If we have the certificate f, we have a flow f with value $\geq k$, hence maxflow value $\geq k$.

Answer for problem B

Algorithm:

// G, s, t, c, k are the input, S is the certificate

Function C(G, s, t, c, k, S)

check if the cut $(S, \overline{S})$ has capacity $\leq k$

If **true**, return **yes**, else return **no**.

Runtime: $O(m)$ by going through all edges of G.

Proof:

If maxflow value $\leq k$, then maxflow mincut theorem shows there is a cut $(S, \overline{S})$ with capacity $\leq k$. Hence, we have the certificate S.

If we have the certificate S, the weak duality of flows and cuts shows that the maxflow value $\leq$ the cut capacity $\leq k$.

Question

Given 2 sequences of **positive** numbers $a_1, \cdots, a_n$, $b_1, \cdots, b_m$.
You are allowed to insert arbitrarily many zeros at any position in both sequences. You want to obtain sequences $\tilde{a}_1, \cdots, \tilde{a}_k$ and $\tilde{b}_1, \cdots, \tilde{b}_k$ with $k \geq \max\{m, n\}$ such that the $\sum \tilde{a}_i \tilde{b}_i$ is maximized.

You only need to output the optimal value $\sum \tilde{a}_i \tilde{b}_i$.

Example:

Input: $a = (1,10,10)$, $b = (10,1,10)$

Output: 200 (for $\tilde{a} = (1,10,0,10)$, $\tilde{b} = (0,10,1,10)$).

Answer

Algorithm:

Let $OPT(i, j)$ be the OPT value for substring $a_1, \cdots, a_i$ and $b_1, \cdots, b_j$

We have

$$OPT(i,j) = \begin{cases} 0 & \text{if } i \le 0 \text{ or } j \le 0 \\ \max \left\{ \begin{array}{c} OPT(i-1, j-1) + a_i b_j, \\ OPT(i-1, j), \\ OPT(i, j-1) \end{array} \right\} & \text{else} \end{cases}.$$

Compute $OPT(n, m)$ using the formula above with memorization.

Runtime: Total time is $O(mn)$ because:

- The recursion only reaches $OPT(i, j)$ for $0 \le i \le n$ and $0 \le j \le m$.
- There is no loop in the recursion since $i + j$ is strictly decreasing.
- Each step takes $O(1)$ time.

Answer

Proof:

Consider the substring $a_1, \cdots, a_i$ and $b_1, \cdots, b_j$.

Case 1) $i \leq 0$ or $j \leq 0$

There is nothing to match except 0. Hence, $OPT(i, j) = 0$.

Case 2) a_i matches with b_j in the optimal matching

We have $OPT(i, j) = OPT(i-1, j-1) + a_i b_j$.

Case 3) a_i matches with 0

We have $OPT(i, j) = OPT(i-1, j) + a_i \cdot 0 = OPT(i-1, j)$.

Case 4) b_j matches with 0

We have $OPT(i, j) = OPT(i, j-1) + 0 \cdot b_j = OPT(i, j-1)$.

Question

Assume P = NP. Given a composite number N. Find a factor a that divides N with $a \neq 1$ and $a \neq N$ in time $\log^{O(1)} N$.

Answer

Algorithm:

Consider the decision problem:

Input: N, l, u

Output: if there is a factor a that divides N such that $l \leq a \leq u$.

Let $A(N, l, u)$ be a poly time algorithm for the problem above.

Call $Find(N, 2, N - 1)$.

Function $Find(N, l, u)$ // Find a factor a that divides N s.t. $l \leq a \leq u$

  If $l = u$, **return** l.

  Let $k = \lfloor (l + u)/2 \rfloor$

  If $A(N, l, k) = True$

    return $Find(N, l, k)$

  else

    return $Find(N, k + 1, u)$

Answer

Runtime and Correctness:

Note that the decision problem is in NP.

The certificate is simply the factor a and checking a divides N takes polytime. (Note that the input size is $\log N$ hence, it is $\log^{O(1)} N$ time).

Using $P = NP$, the decision problem can be decides in $\log^{O(1)} N$ time.

Hence A takes $\log^c N$ time for some c.

Note that $Find$ finds the factor by binary search and it calls A with $\log N$ times in total. Hence, $Find$ takes $\log^{c+1} N$ time.

Each step, we ensure there is some factor a between l and u. This holds initially by the input guarantee N is composite. It is maintained throughout the algorithm due to $A(N, l, k)$. Hence, after $\log N$ steps of divide and conquer, it singles out one integer which is the factor.

Question

Given numbers x_{ij} in an $n \times n$ 2D grid. The numbers on boundary are negative and other numbers are positive.

An element is a peak if it is strictly greater than all of its adjacent neighbors to the left, right, top and bottom.

Give a $O(n \log n)$ time algorithm to find **any** peak element.

Example:

Input:

-1	-1	-1	-1
-1	1	4	-1
-1	3	2	-1
-1	-1	-1	-1

Output: 4 or 3

Answer

Lemma Let $y_i = \max_j x_{ij}$. Suppose i is a peak in y and j is the maximum element among x_{ij}. Then, (i, j) is a peak in x.

Proof:

By definition of peak, $y_{i-1} \le y_i$. Since $y_i = \max_j x_{ij}$, we have

$$x_{i-1,j} \le y_{i-1} \le y_i = x_{ij}.$$

Similarly, $x_{i+1,j} \le x_{ij}$. Finally, since x_{ij} is the maximum among the same i, we have $x_{i,j\pm1} \le x_{ij}$. Hence, (i, j) is a peak in x.

This Lemma shows that it suffices to find a peak in y in $O(\log n)$ time.

Answer

Algorithm:

Call $Find(2, n-1)$

Function $Find(l, u)$

  If $l = u$, **return** l.

  Let $k = \lfloor (l+u)/2 \rfloor$

  If $y_k \leq y_{k+1}$

    return $Find(k+1, u)$

  else

    return $Find(l, k)$

Correctness:

We maintain that

$$y_{l-1} \leq y_l \text{ and } y_u \geq y_{u+1}$$

This is true initially because boundary numbers are negative.

In each step, we recurse according to $y_k \leq y_{k+1}$ or $y_k \geq y_{k+1}$. Hence, this is maintained.

At the end, we have $l = u$ and hence

$$y_{l-1} \leq y_l \text{ and } y_l \geq y_{l+1}$$

Therefore, y_l is a peak.

Runtime: $O(n \log n)$ time because of $O(\log n)$ step, each step involves computing y that takes $O(n)$ time.

Question

Given a graph G with possible negative length. Suppose that there are only k edges with negative length. Given some vertex s, show how to compute the shortest path length from s to every other vertex in $O(mk + kn\log n)$ time.

You can assume there is no negative cycle.

Answer

What do we know?

- Dijkstra takes $O(m + n \log n)$ time, only works with positive length.
- Bellman Ford takes $O(mn)$ time.

Why Bellman Ford is so slow?

- Each step takes $O(m)$ time.
- After k steps, it computes the shortest path distance using k edges.

```
d = Dijkstra(G,c,s,d) {
   Initialize set of explored nodes S ← {s}

   // Maintain distance from s to each vertices in S
   ~~d[s] ← 0~~
   Insert all neighbors v of s into a priority queue with value c_(s,v).

   while (S ≠ V)
   {
      // Pick an edge (u,v) such that u ∈ S and v ∉ S and
      //     d[u] + c_(u,v) is as small as possible.
      u ← delete min element from Q

      Add v to S and define d[v] = min(d[v], d[u] + c_(u,v)).
      Parent(v) ← u.

      foreach (edge e = (v,w) incident to v)
          if (w ∉ S)
             if (w is not in the Q)
                Insert w into Q with value d[v] + c_(v,w)
             else (the key of w > d[v] + c_(v,w))
                Decrease key of v to d[v] + c_(v,w).
}
```

```
SPWithFewNegEdges(G,c,s) {

  Let G⁺ be the set of edges with positive length.
  d[s] ← 0, d[v] ← +∞ for all v ≠ s.
  d = Dijkstra(G⁺,c,s,d)
  for (i = 1,2,⋯k)
  {
    // Bellman Ford step
    d'_v ← min(d_v, min_{(u,v)∈E}(d_u + c_{u,v})) for all v

    // Dijkstra step
    d = Dijkstra(G⁺,c,s,d')
  }}
```

Runtime: $O(mk + nk\log n)$ because it calls **`Dijkstra`** $k+1$ times with k Bellman Ford step.

You can give proof very succinctly and only get deduct very minor points.

Correctness:

Shortest paths only use at most k negative edges.

Induction: After i step, d_v stores the shortest path distance from s to v using at most i negative edges.

Answer

See more details in appendix of https://arxiv.org/abs/2203.03456.

They get almost linear time for negative shortest path.

(Opened for decades)

This question is a subroutine.

This Monday!

Negative-Weight Single-Source Shortest Paths in Almost-linear Time

(Preliminary Version)

Aaron Bernstein* Danupon Nanongkai† Christian Wulff-Nilsen‡

Abstract

We present a randomized algorithm that computes single-source shortest paths (SSSP) in $m^{1+o(1)} \log W$ time when edge weights are integral and can be negative.[1] This essentially resolves classic negative-weight SSSP problem. The previous bounds are $\tilde{O}((m^1 + n^{1.5}) \log W)$ [BLNPSSSW FOCS'20] and $m^{4/3+o(1)} \log W$ [AMV FOCS'20]. In contrast to all recent developments that rely on sophisticated continuous optimization methods and dynamic algorithms, our algorithm is based on a simple graph decomposition and elementary combinatorial tools. In fact, ours is the first combinatorial algorithm for negative-weight SSSP to break through the classic $\tilde{O}(m\sqrt{n} \log W)$ bound from over three decades ago [Gabow and Tarjan SICOMP'89]. Beside being combinatorial, an important feature of our algorithm is in its simplicity: treating our graph decomposition as a black-box, we believe that the reader can reconstruct our algorithm and analysis from our 6-page overview.

Independent result. Independently from our result, the recent major breakthrough by Chen, Kyng, Liu, Peng, Gutenberg, and Sachdeva [CKL+22] achieve an almost-linear time bound for min-cost flow, implying the same bound for our problem. We discuss this result at the end of the introduction.