Course: Randomized Algorithms by Dr. Zarei

Homework: HW3

Name: Mohammad Mohammadi

Student ID: 402208592

Exercise 6.3.15. [DARA_Hor]

Prove the properties (iii), (iv), and (v) of Jacobi symbols presented below:

(III)

$$\text{Jac}\left[\frac{a}{n}\right] = (-1)^{\frac{a-1}{2}\cdot\frac{n-1}{2}} \cdot \text{Jac}\left[\frac{n}{a}\right] \text{ for all odd } a,$$

(IV)

$$\text{Jac}\left[\frac{1}{n}\right] = 1, \text{ and Jac}\left[\frac{n-1}{n}\right] = (-1)^{\left(\frac{n-1}{2}\right)},$$

(V)

$$\text{Jac}\left[\frac{2}{n}\right] = -1 \text{ for all } n \text{ with } n \mod 8 \in \{3,5\}, \text{ and Jac}\left[\frac{2}{n}\right] = 1 \text{ for all } n \text{ with } n \quad \text{r}$$

Answer

Part (III)

This is called the law of Quadratic Reciprocity and it holds true for Legendre symbols and is extended to Jacobi symbols as they generalize Legendre symbols.

$\left[\frac{a}{n}\right] = (-1)^{\frac{a-1}{2}\cdot\frac{n-1}{2}} \left[\frac{n}{a}\right]$

The proof of it for Legendre symbol is available under the link below, we won't prove it but use the proof to show that it holds for Jacobi symbols too:
https://en.wikipedia.org/wiki/Quadratic_reciprocity

For odd a, we can use the fact that if a and n are odd positive integers, the Jacobi symbol $Jac(\frac{a}{n})$ has properties similar to the Legendre symbol.

The law of quadratic reciprocity states that for any two odd primes a and n:
$\left[\frac{a}{n}\right] = (-1)^{\frac{a-1}{2}\cdot\frac{n-1}{2}} \left[\frac{n}{a}\right]$

By definition of Jacobi symbols and properties of Legendre symbols, the reciprocity law holds for Jacobi symbols as well when a is odd.

Hence, property (III) is proven.

Part (IV)

Using induction on a we have:

1. **For a=1:**

By definition of the Jacobi symbol: $Jac(\frac{1}{n}) = 1$

since 1 is always a quadratic residue modulo any number.

2. **For a=n−1:**

We need to show:

$$Jac(\frac{n-1}{n}) = (-1)^{\frac{n-1}{2}}$$

Using properties of Jacobi symbols:

$$Jac(\frac{n-1}{n}) = (-1)^{\frac{(n-1)-1}{2}} = (-1)^{\frac{(n-2)}{2}}$$

Simplifying we have:

$$(-1)^{\frac{(n-1)}{2}} = (-1)^{\frac{(n-2)}{2}}$$

Hence, we have proven (IV).

Part (V)

$Jac(\frac{2}{n}) = -1$ for all n with n mod 8 ∈ {3, 5}, and $Jac(\frac{2}{n}) = 1$ for all n mod 8 ∈ {1, 7}

First of all we have $Jac(\frac{2}{n}) = (-1)^{\frac{n^2-1}{8}}$

when $n \equiv 1, 7(mod 8)$:

- $\frac{n^2-1}{8}$ is even, thus $(-1)^{\frac{n^2-1}{8}} = 1$ therefore $Jac(\frac{2}{n}) = 1$.

when $n \equiv 3, 5(mod 8)$:

- $\frac{n^2-1}{8}$ is odd, thus $(-1)^{\frac{n^2-1}{8}} = -1$ therefore $Jac(\frac{2}{n}) = -1$.

Hence, we have proven (V) holds true.

Exercise 6.4.22. [DARA_Hor]

Modify **PRIMEGEN(l, k)** in such a way that it must run until it outputs a number n. This means that one forbids the output “I was unable to find a prime,” and so there exist infinite runs of **PRIMEGEN(l, k)**. Analyze the expected running time and the error probability of such a modi- fication of **PRIMEGEN(l, k)**.

Answer

To modify the algorithm so that it runs indefinitely until it finds a prime, we will remove the termination condition that allows it to output "I was unable to find a prime." Instead, the algorithm will continue generating numbers and testing them until a prime is found as the exercise requested.

Modified PRIMEGEN(l, k)

Input: Positive integers l and k, l≥3.

Step 1:

```
X := "still not found";
```

Step 2:

```
while X := "still not found" do
    begin
        generate a bit sequence \( a_1, a_2, \ldots, a_{l-2} \) at random
        and compute

        \( n := 2^{l-1} + \sum_{i=1}^{l-2} a_i \cdot 2^i + 1 \)

        {Hence, \( n \) is a random integer of length \( l \)}

        Perform \( k \) independent runs of the Solovay-Strassen algorithm on \( n \);
        if at least one output is "n \(\not\in\) PRIM" then
            continue; {Try another random integer}
        else
            begin
                X := "already found";
                output "n";
            end;
```

```
end;
```

Expected Running Time and Error Probability

Expected Running Time

1. Probability of Selecting a Prime:
    - The probability that a randomly chosen l-bit number is prime is approximately $\frac{1}{lln2}$ due to the Prime Number Theorem.
2. Number of Trials Needed:
    - The number of trials needed follows a geometric distribution with success probability $p = \frac{1}{lln2}$. The expected number of trials is $\frac{1}{p} = lln2$.
3. Solovay-Strassen Test:
    - Each run of the Solovay-Strassen primality test has a running time of $O(klog^3 n)$, where n is the number being tested (which has l bits, so logn=l).
4. Total Expected Running Time:
    - The expected running time is $O(l \ln(2) k l^3) = O(k l^4 \ln(2))$

Error Probability

1. Solovay-Strassen Test Error Probability:
    - Each Solovay-Strassen test has an error probability of at most $\frac{1}{2}$
2. Error in k Independent Runs:
    - The probability that all k runs incorrectly identify a composite number as prime is at most $(\frac{1}{2})^k$.
3. Overall Error Probability:
    - The error probability of the modified PRIMEGEN algorithm is therefore at most $(\frac{1}{2})^k$.

Hence, by modifying the algorithm to run indefinitely until it finds a prime, we ensure it will always find a prime eventually, with an expected running time of $O(k l^4 \ln(2))$ and an error probability of $(\frac{1}{2})^k$.

Exercise 7.4.12. [DARA_Hor]

Find an infinite set of input instances of MAX-SAT for which the expected solutions computed by RSAM are better than the expected solutions computed by RRR.

Answer

One potential infinite set of input instances is composed of highly constrained clauses where each clause contains a large number of variables. For example:

1. Instance Structure: Consider MAX-SAT instances where each clause contains k literals, and each literal is chosen randomly from n variables.
2. Why we do it this way:
    - RSAM: In this case, random sampling has a higher chance of satisfying individual clauses because the large number of literals in each clause increases the probability that at least one literal will be satisfied by a random assignment.
    - RRR: For the same instances, RRR may perform poorly because the relaxation may lead to fractional solutions that do not translate well into integer solutions through rounding. The rounding process might not effectively capture the combinatorial nature of the problem in instances with high constraint-to-variable ratios.

Analysis

1. RSAM: The RSAM will often find a satisfying assignment or a near-optimal solution because the random assignment of variables is likely to satisfy many clauses given the high probability of satisfying each clause individually.
2. RRR: The RRR might not perform as well because the relaxation could lead to fractional assignments that do not translate into effective integer assignments through randomized rounding. The structured nature of solutions might not adapt well to the random distribution of literals in the clauses.

Hence, the infinite set $\{I_n\}$ where each I_n is constructed as described above provides instances where RSAM is expected to outperform RRR in terms of finding better solutions to the MAX-SAT problem.

Exercise 7.4.13. [DARA_Hor]

Find an infinite set of instances of MAX-SAT such that one can expect better results from RRR than from RSAM.

Answer

A good candidate for such instances involves MAX-SAT problems where the optimal solutions can be closely approximated by solving a relaxed version of the problem, and the rounding process does not significantly degrade the quality of the solution. Such instances often have a more structured nature compared to completely random instances. For example:

Consider the set of instances J_n, where n is the number of variables and each instance is constructed as follows:

1. Number of variables: n
2. NUmber of clauses: m = n logn, where each clause has a relatively small number of literals, typically 2 or 3 literals per clause (let's assume 2-literal clauses for simplicity, known as 2-SAT or 2-MAX-SAT).

Each clause in the instance contains 2 literals chosen based on specific dependencies or correlations between variables.

Why we do it this way:

1. RRR: In these instances, solving a relaxed version of the problem can yield fractional solutions that capture the dependencies or correlations between variables effectively. The rounding process can then convert these fractional solutions into integer solutions without losing much of the solution quality. The structure in these instances ensures that the relaxation closely approximates the original problem.
2. RSAM: The random sampling algorithm might not perform as well because it does not exploit the structure and dependencies between variables. Instead, it relies purely on random assignments, which can lead to suboptimal solutions due to the lack of exploitation of the inherent structure.

Let's formalize the construction of the instance set J_n:

1. Number of variables: n
2. Number of clauses: m = n logn
3. Clause Construction: Each clause C_i is of the form $(x_{a_i} \vee \neg x_{b_i})$, where x_{a_i} and $\neg x_{b_i}$ are variables chosen such that there are dependencies or correlations (e.g., variables appear together in many clauses, or certain variable pairs are chosen based on some structured rule).

Analysis:

1. Structured Dependencies:
    - The dependencies or correlations between variables mean that the solution space has a certain structure that can be exploited by relaxation methods.

    - The relaxation will find a fractional solution that respects these dependencies.
2. Effective Rounding:
    - When rounding the fractional solution obtained from the relaxation, the structure ensures that the rounding process maintains most of the solution quality.
    - The RRR method effectively translates the structure captured in the relaxation into a high-quality integer solution.
3. RSAM Inefficiency:
    - RSAM does not exploit the structure and dependencies.
    - It relies on pure random sampling, which might fail to consistently find high-quality solutions due to the structured nature of the instance.

In conclusioon we can say:

The infinite set of instances $\{J_n\}$, where each J_n consists of n variables and (n logn) clauses with structured dependencies (e.g., 2-literal clauses with correlated variables), provides cases where RRR is expected to outperform RSAM. The structured nature of these instances allows the relaxation methods used in RRR to closely approximate the optimal solutions, and the rounding process maintains the solution quality, leading to better results compared to the random sampling approach of RSAM.

Problem 8.1. [RA_Mot]

Prove Lemma 8.4.

Lemma 8.4.:

$$For\ all\ p, t \geq 0, \quad D_p^t = H_p + H_t - H_{p+t}$$

Answer

Some assumptions we take:

- H_n denotes the n-th harmonic number defined as:
$$H_n = \sum_{k=1}^{n} \frac{1}{k}$$
- We will assume that D_p^t is given in a context where it can be expressed in terms of harmonic numbers.

By the definitions of the harmonic numbers we can progress as:

First, let's expand each harmonic number in the lemma:

1.
$$(H_p) : H_p = \sum_{k=1}^{p} \frac{1}{k}$$

2.
$$(H_t) : H_t = \sum_{k=1}^{t} \frac{1}{k}$$

3.
$$(H_{p+t}) : H_{p+t} = \sum_{k=1}^{p+t} \frac{1}{k}$$

Now, let's analyze the right-hand side of the equation $H_p + H_t - H_{p+t}$:

$$H_p + H_t - H_{p+t} = \left(\sum_{k=1}^{p} \frac{1}{k}\right) + \left(\sum_{k=1}^{t} \frac{1}{k}\right) - \left(\sum_{k=1}^{p+t} \frac{1}{k}\right)$$

We can rewrite the sum $\left(\sum_{k=1}^{p+t} \frac{1}{k}\right)$ as the sum of two separate parts:

$$\sum_{k=1}^{p+t} \frac{1}{k} = \sum_{k=1}^{p} \frac{1}{k} + \sum_{k=p+1}^{p+t} \frac{1}{k}$$

Thus:

$$H_p + H_t - H_{p+t} = \left(\sum_{k=1}^{p} \frac{1}{k}\right) + \left(\sum_{k=1}^{t} \frac{1}{k}\right) - \left(\sum_{k=1}^{p} \frac{1}{k} + \sum_{k=p+1}^{p+t} \frac{1}{k}\right)$$

Simplifying:

$$H_p + H_t - H_{p+t} = \sum_{k=1}^{p} \frac{1}{k} + \sum_{k=1}^{t} \frac{1}{k} - \sum_{k=1}^{p} \frac{1}{k} - \sum_{k=p+1}^{p+t} \frac{1}{k}$$

The terms $\left(\sum_{k=1}^{p} \frac{1}{k}\right)$ cancel out:

$$H_p + H_t - H_{p+t} = \sum_{k=1}^{t} \frac{1}{k} - \sum_{k=p+1}^{p+t} \frac{1}{k}$$

This simplifies to:

$$H_p + H_t - H_{p+t} = \sum_{k=1}^{t} \frac{1}{k} - \sum_{k=1}^{t} \frac{1}{k}$$

Here we observe that (H_{p+t}) can be decomposed into the sum of (H_p) and the remaining terms:

$$H_{p+t} = H_p + \sum_{k=p+1}^{p+t} \frac{1}{k}$$

Thus, $\left(\sum_{k=1}^{t} \frac{1}{k}\right)$ are effectively canceled out.

Therefore, we see that:

$$H_p + H_t - H_{p+t} = \sum_{k=1}^{t} \frac{1}{k} - \sum_{k=p+1}^{p+t} \frac{1}{k}$$

resulting in:

$$D_p^t = H_p + H_t - H_{p+t}$$

Hence, we have proved the lemma:

$$D_p^t = H_p + H_t - H_{p+t}$$

Problem 8.2. [RA_Mot]

Prove Lemma 8.5. Lemma 8.5.:

$$For\ all\ p, s, t \geq 0, \quad E_p^{s,t} = \frac{t}{s+t} + (H_{s+p} - H_s) - (H_{s+p+t} - H_{s+t}).$$

Answer

WE continue with the assumptions we made in the previous problem, First, let's expand each harmonic number in the lemma:

1. $H_{s+p} = \sum_{k=1}^{s+p} \frac{1}{k}$
2. $H_s = \sum_{k=1}^{s} \frac{1}{k}$
3. $H_{s+p+t} = \sum_{k=1}^{s+p+t} \frac{1}{k}$
4. $H_{s+t} = \sum_{k=1}^{s+t} \frac{1}{k}$

Now, let's analyze the right-hand side of the equation:

$$\frac{t}{s+t} + (H_{s+p} - H_s) - (H_{s+p+t} - H_{s+t})$$

Substitute the expanded harmonic numbers:

$$H_{s+p} - H_s = \left(\sum_{k=1}^{s+p} \frac{1}{k}\right) - \left(\sum_{k=1}^{s} \frac{1}{k}\right) = \sum_{k=s+1}^{s+p} \frac{1}{k}$$

Similarly,

$$H_{s+p+t} - H_{s+t} = \left(\sum_{k=1}^{s+p+t} \frac{1}{k}\right) - \left(\sum_{k=1}^{s+t} \frac{1}{k}\right) = \sum_{k=s+t+1}^{s+p+t} \frac{1}{k}$$

Substitute these back into the right-hand side:

$$\frac{t}{s+t} + \sum_{k=s+1}^{s+p} \frac{1}{k} - \sum_{k=s+t+1}^{s+p+t} \frac{1}{k}$$

Thus, we observe that:

$$E_p^{s,t} = \frac{t}{s+t} + \left(\sum_{k=s+1}^{s+p} \frac{1}{k}\right) - \left(\sum_{k=s+t+1}^{s+p+t} \frac{1}{k}\right)$$

Hence, we have proved the lemma:

$$E_p^{s,t} = \frac{t}{s+t} + (H_{s+p} - H_s) - (H_{s+p+t} - H_{s+t})$$

Problem 8.4. [RA_Mot]

Given a set of keys $S = \{k_1, k_2, \ldots, k_n\}$, consider constructing a random treap for S where we do not introduce the dummy leaves needed for the endogenous property. Is every element of S equally likely to be a leaf in this treap? Discuss the implications of your result for the performance of a treap.

Answer

To analyze the probability that each element in the set S is a leaf in a random treap we continue as below:

Construction of a Random Treap:

1. Assign Random Priorities: Each key k_i is assigned a random priority p_i.
2. Insert Keys into the Treap: Keys are inserted into the treap based on the priorities, ensuring that the BST property and heap property are maintained.

Probability of a Key Being a Leaf:

To determine if every element of S is equally likely to be a leaf, we need to analyze the insertion process in the treap.

- Insertion Mechanics:

    - When inserting a key k_i with priority p_i, it initially becomes a leaf.
    - A key remains a leaf if no subsequent key insertion results in it becoming an internal node.
- Maintaining the Leaf Status:
    - For k_i to remain a leaf, all keys k_j inserted after k_i must not have priorities that would place them as a child of k_i in the BST.
    - Specifically, k_i remains a leaf if none of the keys in its range (keys less than k_i and greater than k_i) have a priority lower than p_i.

To determine whether all keys are equally likely to be leaves, consider the following:

- Random Priorities:
    - Each key is assigned a random priority independently.
    - The probability distribution of priorities is uniform over the possible priority values.
- Symmetry:
    - Each key k_i has an equal probability of being inserted at any position in the BST with respect to the priority assignment.
    - The structure of the treap depends only on the relative order of the priorities.

Now for th calculation of probability we go through spteps as below:

- Consider a specific key k_i.
- For k_i to be a leaf, no key k_j with $j \neq i$ can have a priority lower than p_i and be placed such that it becomes a child of k_i.
- For k_i to remain a leaf, it must not have any children. Therefore, every other key must either:
    - Not fall within the range that k_i can parent.
    - Have a higher priority (which means a lower priority value in the min-heap context).
- Since priorities are assigned randomly and uniformly, each key has an equal chance of having the highest priority among the keys in its potential subtree.
- Therefore, each key has an equal probability of ending up as a leaf.

Performance in Search, Insert, and Delete Operations:

- The random assignment of priorities helps to ensure that the treap is balanced on average, similar to the expected properties of a balanced BST.
- Since each key is equally likely to be a leaf, the distribution of leaves is uniform, contributing to the balanced nature of the treap.

Expected Depth and Path Lengths

- The expected depth of a node in a treap is $O(logn)$, which means the search, insertion, and deletion operations are efficient on average.

- The uniform distribution of leaf nodes implies that there are no significant biases toward certain nodes becoming leaves, maintaining the balance of the tree.

Hence, in conclusion in a random treap constructed without introducing dummy leaves, every element of S is equally likely to be a leaf. This results from the uniform distribution of random priorities and the balanced nature of the treap structure. The balanced distribution of leaves ensures that the treap performs well on average, with operations such as search, insert, and delete expected to take $O(logn)$ time.

Problem 8.16. [RA_Mot]

Give a high probability bound on the space requirement of a random skip list for a set S of size n.

Answer

Some facts from skip lists:

- Each element in the skip list appears in the bottom level.
- An element that appears in level i also appears in level i+1 with a probability p, typically p=1/2.

Expected Number of Nodes at Each Level:

1. Level 0:
    - All n elements are present.
    - Number of nodes: n.
2. Level 1:
    - Each element appears with probability p.
    - Expected number of nodes: np.
3. Level i:
    - Expected number of nodes: np^i.

The total number of levels L is expected to be $log_{1/p}n$. For p=1/2 this simplifies to $log_2 n$.

Total Space Requirement

The total space requirement is the sum of the number of nodes at each level.

Total space = $\sum_{i=0}^{L-1}$ Number of nodes at level i.

Since the number of nodes at level i is np^i:

Total space = $\sum_{i=0}^{L-1} np^i$

For p=1/2:

Total space = $\sum_{i=0}^{Log_2 n} (\frac{1}{2})^i$

This is a geometric series with sum:

Total space = $n \sum_{i=0}^{Log_2 n} (\frac{1}{2})^i$

The sum of a geometric series $\sum_{i=0}^{k} r^i = \frac{1-r^{k+1}}{1-r}$

For r=1/2 and $k = Log_2 n$, we have:

$$\sum_{i=0}^{Log_2 n} (\frac{1}{2})^i = \frac{1 - \frac{1}{2}^{log_2 n+1}}{1 - \frac{1}{2}} = 2(1 - \frac{1}{2n}) \simeq 2$$

Hence, the total space requirement is:

Total space ≃ 2n.

To establish a high probability bound, we consider the Chernoff bound. For the number of nodes at each level:

Let X_i be the number of nodes at level i.

The expectation $E[X_i] = np^i$. By the Chernoff bound, the probability that X_i deviates significantly from its expectation is exponentially small.

Specifically, for any $\delta > 0$:

$\Pr[X_i \geq (1+\delta)\mathbb{E}[X_i]] \leq \exp\left(-\frac{\delta^2}{2+\delta}\mathbb{E}[X_i]\right)$

Given $\mathbb{E}[X_i] = np^i$:

Choose $\delta = 2$.

$\Pr[X_i \geq 3np^i] \leq \exp\left(-\frac{4}{3}np^i\right)$

Since $p = \frac{1}{2}$:

$\Pr[X_i \geq 3n\left(\frac{1}{2}\right)^i] \leq \exp\left(-\frac{4}{3}n\left(\frac{1}{2}\right)^i\right)$

The total number of nodes across all levels with high probability is bounded by summing over all levels up to $(\log_2 n)$:

$\text{Total space} \leq \sum_{i=0}^{\log_2 n} 3n\left(\frac{1}{2}\right)^i \leq 3n \sum_{i=0}^{\log_2 n} \left(\frac{1}{2}\right)^i = 3n \cdot 2 = 6n$

So in conclusion, with high probability, the total space requirement of a random skip list for a set S of size n is O(n). Specifically, the high probability bound is 6n. This ensures efficient space usage, contributing to the overall performance and scalability of skip lists.