Algorithm Design: Homework #5

Due on June 26th, 2020 at 11:59pm

Professor Rui Fan

Tianyuan Wu
63305667

Problem 1

Solution

a) It's obviously that if we choose k times $X = X+1$ and $n-k$ times $X = X-1$, then $X = k-(n-k) = 2k-n$. So, if finally $X = i$, then $k = \frac{n+i}{2}$, and $n-k = \frac{n-i}{2}$ Hence,

$$\begin{aligned}\Pr[X=i] &= \binom{n}{(n+i)/2}(\frac{1}{2})^k(\frac{1}{2})^{n-k} \\ &= \binom{n}{(n+i)/2}(\frac{1}{2})^n\end{aligned}$$

Notice that the symmetry of this equation, we know that $\binom{n}{m} = \binom{n}{n-m}$, hence $Pr[X=k] = \Pr[X = -k]$.

Thus we can calculate $\mathbb{E}[X]$:

$$\begin{aligned}\mathbb{E}[X] &= \sum_{i=0}^{n} i\Pr[x=i] \\ &= \sum_{i=-n/2}^{n/2} i\binom{n}{(n+i)/2}(\frac{1}{2})^n \\ &= 0\end{aligned}$$

b) For even n, we know that

$$\begin{aligned}\Pr[X=0] &= \binom{n}{n/2}(\frac{1}{2})^n \\ &= \frac{n!}{((n/2)!)^2}(\frac{1}{2})^n \\ &= \frac{(\frac{n}{e})^n\sqrt{2\pi n}}{\pi n(\frac{n}{2e})^n}(\frac{1}{2})^n \\ &= \sqrt{\frac{2}{\pi n}}\frac{n^n}{e^n}\frac{2^n e^n}{n^n}(\frac{1}{2})^n \\ &= \sqrt{\frac{2}{\pi n}}\end{aligned}$$

And for odd n's, X can never be 0, so $\Pr[X=0] = 0$.

Problem 2

Solution

The algorithm is, for each time, choose a vertex that is not been colored, then generate a uniform distributed random variable $X \sim Unif(0,3)$. Then if $0 < X < 1$, assign this vertex to color 1; if $1 < X < 2$, assign it to color 2, else assign it to color 3. (i.e. color this vertex to 3 colors with same probability), until all vertices are colored.

Then, we can prove that, for each adjacent pair of vertices V_i and V_j, the probability of they are assigned to same color is $P = 1 - \binom{3}{1}(\frac{1}{3})^2 = \frac{2}{3}$. Hence the expectation of satisfied edges found by this algorithm satisfies: $c_{found} > \frac{2}{3}c_{total}$, where c_{total} is the total number of edges in the graph. And it's obviously that $c^* < c_{total}$, so $c_{found} \geq \frac{2}{3}c^*$.

Problem 3

Solution

1. It's obviously that $E[X_i]$ is equal to the probability of picking an 1 in the array, So

$$\mathbb{E}[X_i] = \frac{T}{n}$$

2.

$$\begin{aligned} Var[X_i] &= Pr[X_i = 0](0 - E[X_i])^2 + Pr[X_i = 1](1 - E[X_i])^2 \\ &= (1 - \frac{T}{n})(\frac{T}{n})^2 + \frac{T}{n}(1 - \frac{T}{n})^2 \\ &= (1 - \frac{T}{n})(\frac{T}{n}) \end{aligned}$$

3. By the linearity of expectation:

$$\begin{aligned} \mathbb{E}[Y] &= \mathbb{E}[\frac{n}{s}\sum_{i=1}^{n} X_i] \\ &= \frac{n}{s}\sum_{i=1}^{n} \mathbb{E}[X_i] \\ &= s \cdot \frac{n}{s} \cdot \frac{T}{n} \\ &= T \end{aligned}$$

4. By the linearity of variance of i.i.d variables:

$$Var[X_1 + X_2 \ldots + X_s] = s \cdot Var[X_1]$$

and

$$Var[cX] = c^2 Var[X]$$

We have:

$$Var[Y] = \frac{n^2}{s^2} \cdot s \cdot Var[X_i] = \frac{n^2}{s} Var[X_i]$$

Hence,

$$Var[Y] = \frac{n^2}{s}(1 - \frac{T}{n})(\frac{T}{n}) = \frac{nT}{s}(1 - \frac{T}{n})$$

It takes its maximum value at $T = \frac{n}{2}$, so if T is at about $\frac{n}{2}$, the variance is large. So, the performance is not good (i.e. it's not a good estimator).

Problem 4

Solution

The algorithm is:

Algorithm 1 Maximum 3D matching

```
Initialize S = {}
while T are all non-empty do
    Randomly choose a triple (X_i, Y_j, Z_k) from T
    Add (X_i, Y_j, Z_k) to S
    Delete all triples related to (X_i, Y_j, Z_k) in T
end while
```

Notice that the matching found by this algorithm is not a subset of any other matching in T, for we found add triples until no triples can be added.

For any 2 different subset S_1 and S_2 found by this algorithm, we'll prove that $|S_1| \leq 3|S_2|$.

First notice that: each triple (x_i, y_j, z_k) in $S_2 \setminus S_1$ can be adjacent to at most 3 edges in $S_1 \setminus S_2$. For $S_1 \setminus S_2$ at most contains $(x_i, \_, \_)$, $(\_, y_j, \_)$, $(\_, \_, z_k)$. And each edge in $S_1 \setminus S_2$ is adjacent to an edge in $S_2 \setminus S_1$. since S_2 is a maximal matching.Hence

$$|S_1 \setminus S_2| \leq 3|S_2 \setminus S_1|$$

Thus we have:

$$|S_1| = |S_1 \cap S_2| + |S_1 \setminus S_2| \leq 3|S_1 \cap S_2| + 3|S_2 \setminus S_1| = 3|S_2|$$

So we proved $|S_1| \leq 3|S_2|$. By $|S_1|, |S_2| \leq |T|$, $|S_1| \leq 3|S_2|$ and the size of maximum 3D matching is also less than the size of T (i.e. $|M| \leq |T|$), we can observe for any set S found by our algorithm:

$$|S| \geq \frac{1}{3}|M|$$

Hence we can find 3 dimensional matching of size at least 1/3 times the maximum possible size by this algorithm. For the time complexity, we choose at most $O(n)$ times, and for each choice, at most delete $O(n)$ triples. Hence the time complexity is:

$$T(n) = O(n^2)$$

Problem 5

Solution

a) Let $v \in T$. If $v \notin S$, we can observe that there must exists a vertex v' which is the neighbor of v. And v' is selected by the greedy algorithm and v is removed. Then, we must have that: $w(v') \geq w(v)$, otherwise the greedy algorithm will choose v instead of v'. Thus, for each node $v \in T$, either $v \in S$, or there is a node$v' \in S$ so that $w(v) < w(v')$ and (v, v') is an edge of G.

b) Let $C = S \cap T$, and $S' = S \setminus C$, $T' = T \setminus C$. By (a), we know that for each node $v \in T'$, there is a node $v' \in S$ so that $w(v) < w(v')$ and (v, v') is an edge of G. We use $v' \in S'$ to cover for v in T', and any such v' need to cover at most 4 neighbors of $v \in T'$ where $w(v) \leq w(v')$. Thus we have

Problem 5 continued on next page...

$w(S') \leq 4w(T')$. Hence,

$$\begin{aligned} w(S) &= w(S') + w(C) \\ &\geq w(C) + \frac{1}{4}w(T') \\ &\geq \frac{1}{4}(w(T') + w(C)) \\ &= \frac{1}{4}w(T) \end{aligned}$$

Hence, we've proved the "heaviest-first" algorithm returns an independent set of totalweight at least 1/4 times the maximum total weight of any independent set in G.

Problem 6

Solution

a) If $w_1 = 1$, $w_2 = 2$, $w_3 = 3$, $w_4 = 4$, $w_5 = 5$, $w_6 = 6$, and $K = 7$, then the minimum possible number of trucks is 3. Because we can let first truck contain w_1, w_6, second truck contain w_2, w_5, third truck contain w_3, w_4. But by this algorithm, we need 4 trucks: first contains w_1, w_2, w_3, second contains w_4, third contains w_5, fourth contains w_6, which is not optimal.

b) Let T_i denotes the items in truck i by our greedy approach, and W_i denotes the total weight of items in truck i (i.e. $W_i = \sum_{a \in T_i} w_a$). Then we can observe that:

$$W_i + W_{i-1} > K$$

for any i. Because if $W_i + W_{i-1} \leq K$, our greedy algorithm will merge T_i and T_{i-1} together.
Then we discuss following 2 cases:
If $N = 2m$ (N is even) for some integer m:

$$\sum_{j=1}^{N} W_j = \sum_{j=1}^{m}(W_{2j} + W_{2j-1}) > Km$$

and $N = 2m + 1$ (N is odd) for some integer m:

$$\sum_{j=1}^{N} W_j = \sum_{j=1}^{m}(W_{2j} + W_{2j-1}) + W_{2m+1} > Km$$

For $m \geq \frac{N-1}{2}$, we can observe

$$\sum_{i=1}^{n} w_i = \sum_{j=1}^{N} W_j > Km \geq \frac{K(N-1)}{2}$$

And for the minimum number of trucks N^* we have:

$$\sum_{i=1}^{n} w_n \leq KN^*$$

By inequalities shown above, we know

$$\frac{K(N-1)}{2} < KN^*$$
$$N < 2N^* + 1$$

Hence,

$$N \leq 2N^*$$

Problem 7

Solution

a) Let $S = \{1, 2, 3, 101\}$, and $B = 102$. then the optimal solution is choosing $S = \{1, 101\}$, but the solution given by this algorithm is $S = \{1, 2, 3\}$. then for $6 < 102$, this solution is less than half of some other solution.

b) The algorithm is: That is a greedy algorithm which picks the one with maximum weight that can be

Algorithm 2 Maximum subset sum

```
Initialize S = {}, i = n, C = B
Sort {a_1, a_2, ..., a_n} to descending order
while i ≠ 0 do
    if a_i < C then
        C = C − a_i
        Add a_i to S
    end if
    i = i − 1
end while
```

added to the subset.
Proof:
If subset S found by our algorithm has total weight W, and W^* denotes the optimal solution (i.e. $W = \sum_{a \in S} w_a$). We hypothesis that $W < \frac{1}{2}W^*$. By the hypothesis, we have $\exists a \in (A \setminus S), w_a \geq \frac{1}{2}B$. But the items in the set satisfies $w < \frac{1}{2}B$, which means we didn't pick the item with maximum weight. By this contradiction, we know that:

$$W = \sum_{a \in S} w_a \geq \frac{1}{2}B$$

And the total weight of the optimal solution S^* is:

$$W_{S^*} = \sum_{a \in S^*} w_a \leq B$$

Hence, we have:

$$W \geq \frac{1}{2}W_{S^*}$$

Thus we found an algorithm which returns a feasible set $S \subseteq A$ whose total sum is at least half as large as the maximum total sum of any feasible set $S' \subseteq A$.
The time complexity is $O(nlogn)$, for the sort takes $O(nlogn)$ time, and assignment takes $O(n)$ time.