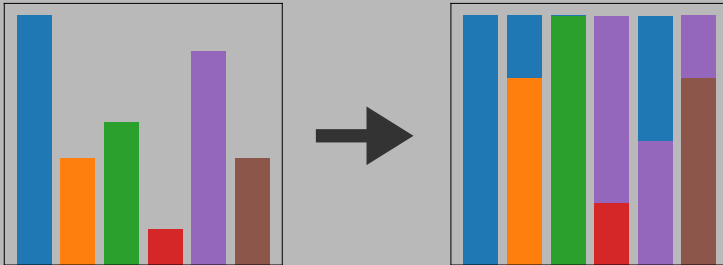


Alias Table sampling

Basile Fraboni

GDL Origami



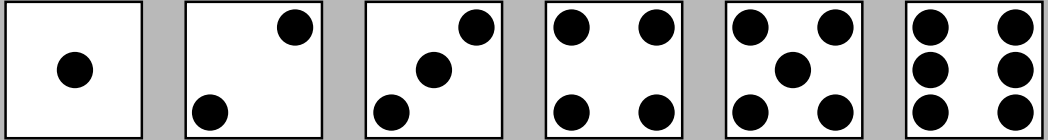
What is an Alias Table ?

What is an Alias Table ?

- A structure to sample discrete distributions

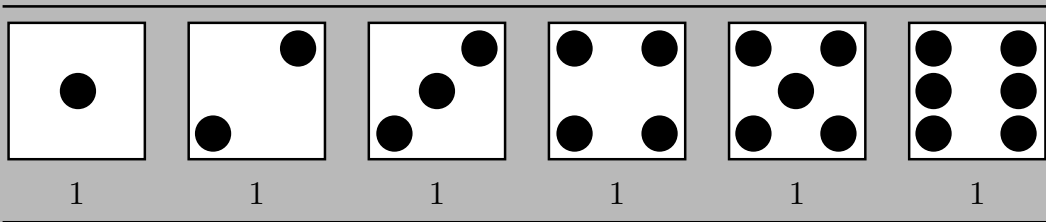
What is an Alias Table ?

- A structure to sample discrete distributions



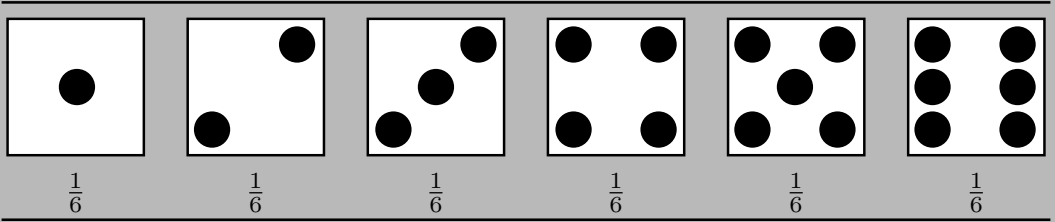
What is an Alias Table ?

- A structure to sample discrete distributions



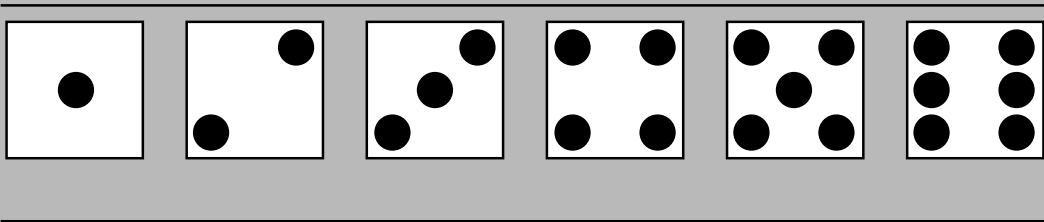
What is an Alias Table ?

- A structure to sample discrete distributions



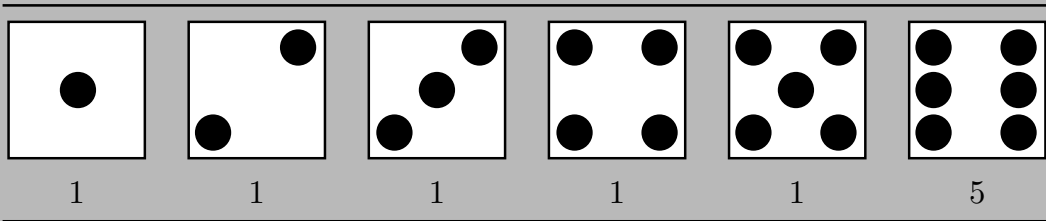
What is an Alias Table ?

- A structure to sample discrete distributions



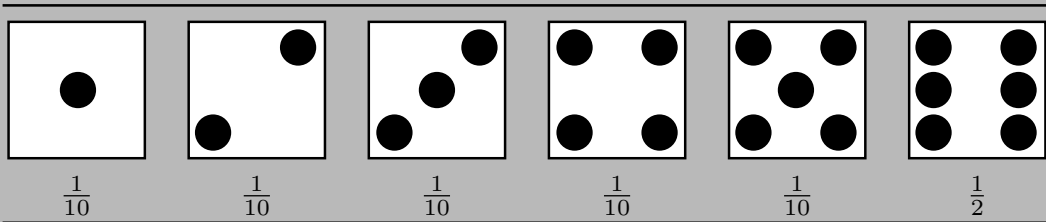
What is an Alias Table ?

- A structure to sample discrete distributions



What is an Alias Table ?

- A structure to sample discrete distributions



What is an Alias Table ?

- A structure to sample discrete distributions

A	B	C	D	E	...
w_A	w_B	w_C	w_D	w_E	...

What is an Alias Table ?

- A structure to sample discrete distributions

A	B	C	D	E	...
w_A	w_B	w_C	w_D	w_E	...

- importance sampling

What is an Alias Table ?

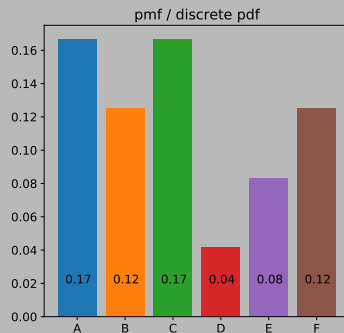
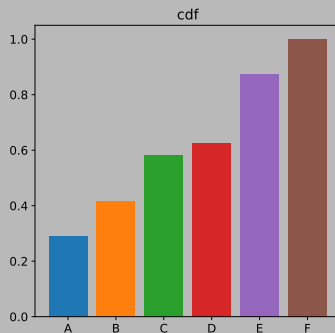
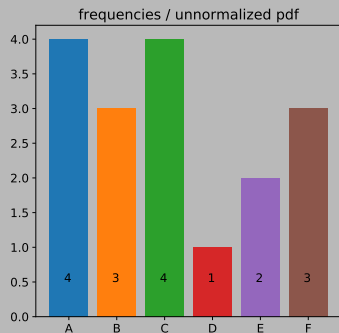
- A structure to sample discrete distributions

A	B	C	D	E	...
w_A	w_B	w_C	w_D	w_E	...

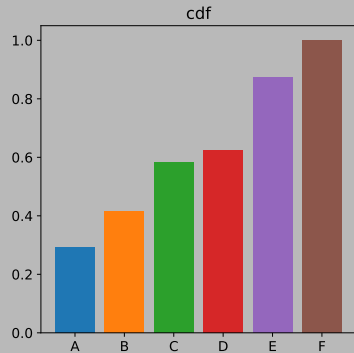
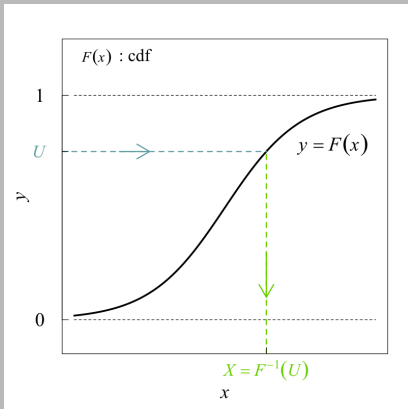
- importance sampling
- simulations, data analysis, machine learning, statistics, etc

Discrete CDF

A	B	C	D	E	F
7	3	4	1	6	3

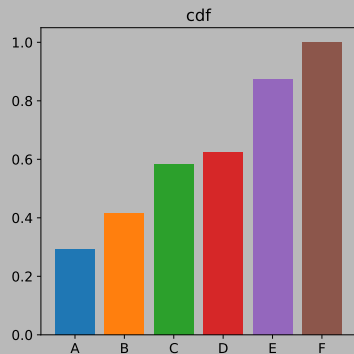
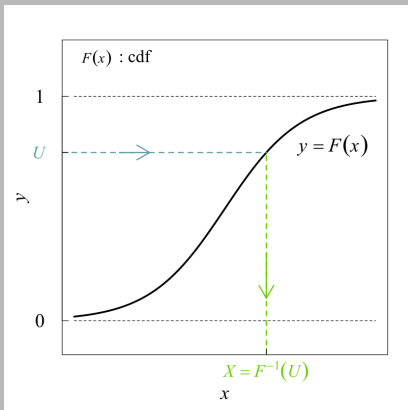


Discrete CDF inversion



Example image from Wikimedia

Discrete CDF inversion



- binary search $O(\log n)$

Example image from Wikimedia

Discrete CDF implementation

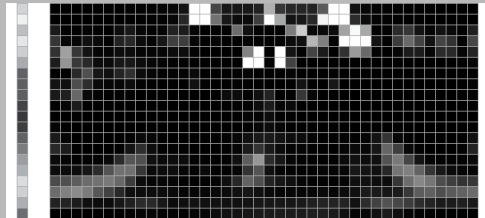
```
struct DiscreteCDF
{
    int n;
    float total;
    std::vector<float> cdf;

    DiscreteCDF(const std::vector<float>& values) : n(values.size()), total(0), cdf(n)
    {
        for(int i = 0; i < n; ++i)
        {
            total += values[i];
            cdf[i] = total;
        }
    }

    int sample(const float u)
    {
        const float value = u * total;
        int p = 0, q = n - 1;
        while(p < q)
        {
            int m = (p+q) / 2;
            if(cdf[m] < value)
                p = m + 1;
            else
                q = m;
        }
        return p;
    }
};
```

Discrete CDF

- rendering: sampling area (lights), images (environment maps), volumes, etc



Example images from PBRT [Pharr et al. 2016]

Discrete CDF

Discrete CDF	
construction	$O(n)$
sampling	$O(\log n)$

What is an Alias Table ?

- A structure to sample discrete distributions

A	B	C	D	E	...
w_A	w_B	w_C	w_D	w_E	...

- **in constant time !**

What is an Alias Table ?

Regained attention recently:

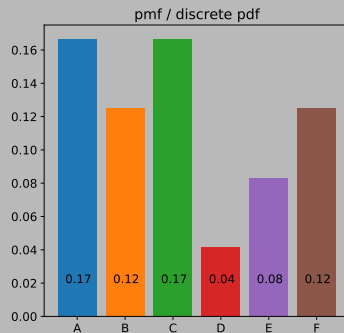
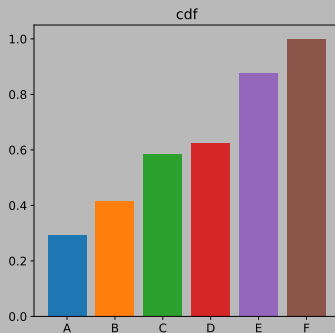
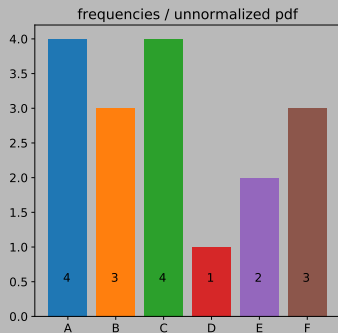
- **New fast method for generating discrete random numbers with arbitrary frequency distributions**, *A. J. Walker*, 1974
- **A linear algorithm for generating random numbers with a given distribution**, *M. D. Vose*, Software Engineering, 1991
- **Parallel Weighted Random Sampling**, *L. Hübschle-Schneider and P. Sanders*, 2019 - 2021
- **Weighted Random Sampling on GPUs**, *H.-P. Lehmann, L. Hübschle-Schneider and P. Sanders*, 2021
- **The alias method for sampling discrete distributions**, *C. Wyman*, Ray tracing Gems 2, Chapter 21, 2021
- used in the implementation of the ReSTIR algorithm [Bitterli et al. 2020]

What is an Alias Table ?

	Discrete CDF	Alias [Walker]	Alias [Vose]	Alias [Hübschle]
construction	$O(n)$	$O(n^2)$	$O(n)$	$O(n)$
sampling	$O(\log n)$	$O(1)$	$O(1)$	$O(1)$

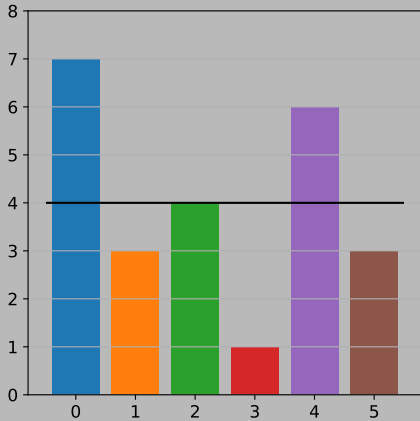
Alias Table Construction

A	B	C	D	E	F
7	3	4	1	6	3



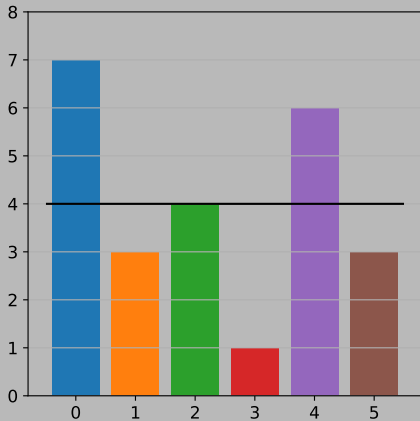
Integers are used for the example, but the construction applies for real weights – frequencies.

Alias Table Construction

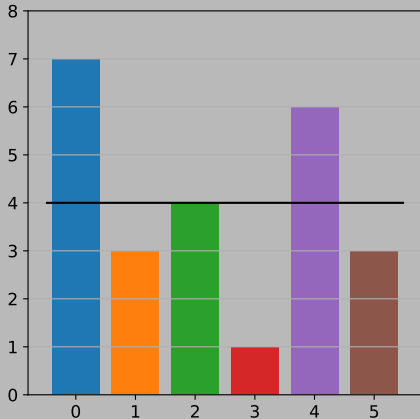


Alias Table Construction

- compute mean

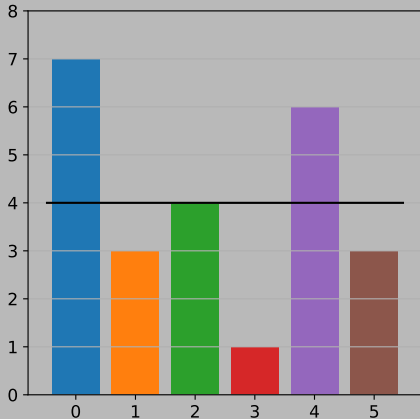


Alias Table Construction



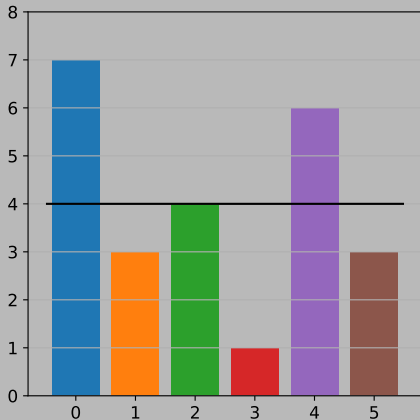
- compute mean
- **large items** $> \text{mean}$
- **small items** $\leq \text{mean}$

Alias Table Construction



- compute mean
- **large items** $> \text{mean}$
- **small items** $\leq \text{mean}$
- split large items and share residuals

Alias Table Construction



- compute mean
- **large items** > mean
- **small items** <= mean
- split large items and share residuals

■ **Algorithm 2** A sweeping algorithm for building alias tables.

Input: $\langle w_1, \dots, w_n \rangle \in \mathbb{R}^n$ the weights of the n input items

assume sentinel items $w_{n+1} = \infty$ and $w_{n+2} = 0$ to avoid some special case treatments

Output: b , an alias table consisting of n pairs (w, a) of (partial) weight w and alias a

1 **Function** sweepingAliasTable($\langle w_1, \dots, w_n \rangle$)

2 $W := \sum_{i=1}^n w_i$ — total weight

3 $i := \min \{k > 0 : w_k \leq W/n\}$ — first light item

4 $j := \min \{k > 0 : w_k > W/n\}$ — first heavy item

5 $w := w_j$ — current heavy item

6 **if** $j = n + 1$ **then** $\forall k = 1..n : b[k].p = w_k; b[k].a = k;$ — All weights are equal

7 **while** $j \leq n$ **do**

8 **if** $w > W/n$ **then** — Pack a light bucket.

9 $b[i].w := w_i$ — Item i completely fits here.

10 $b[i].a := j$ — Item j fills the remainder of bucket i .

11 $w := (w + w_i) - W/n$ — Update residual weight of item j .

12 $i := \min \{k > i : w_k \leq W/n\}$ — next light item

13 **else** — Pack a heavy bucket.

14 $b[j].w := w$ — Now item j completely fits here.

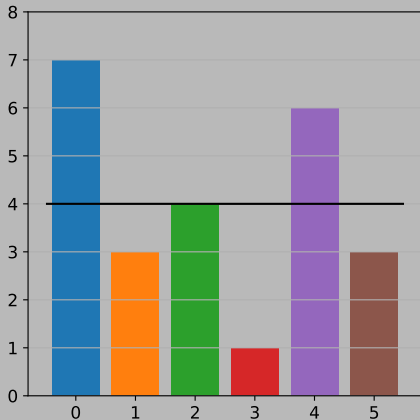
15 $j' := \min \{k > j : w_k > W/n\}$ — next heavy item

16 $b[j].a := j'; j := j'$ — Proceed with item j'

17 $w := (w + w_{j'}) - W/n$ — Compute residual weight avoiding cancellation issues

[Hübschle et al. 2019]

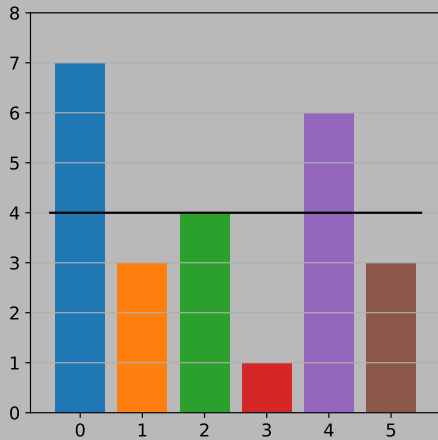
Alias Table Construction



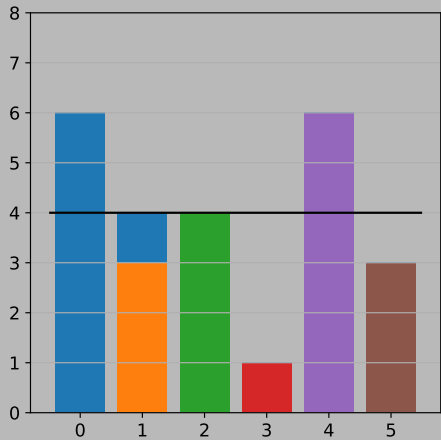
Required operations:

- find next small item
- find next large item
- pack small item
- pack large item

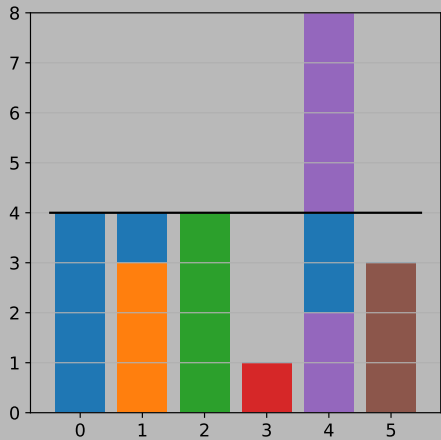
Alias Table Construction



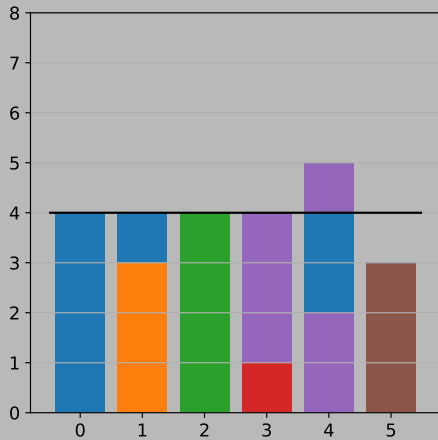
Alias Table Construction



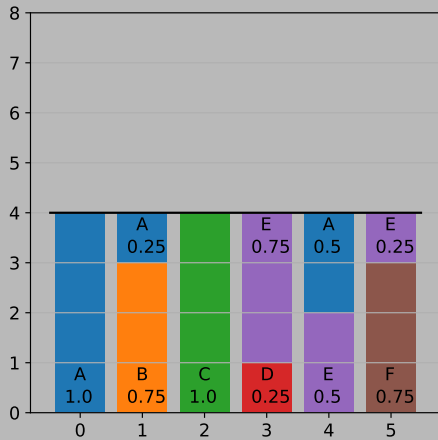
Alias Table Construction



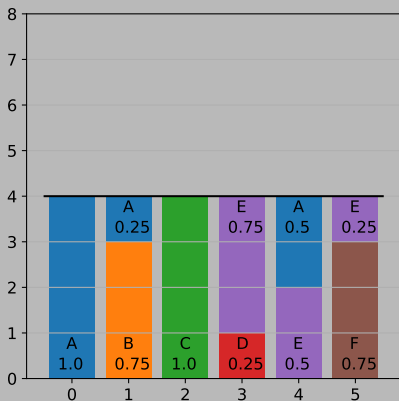
Alias Table Construction



Alias Table Construction

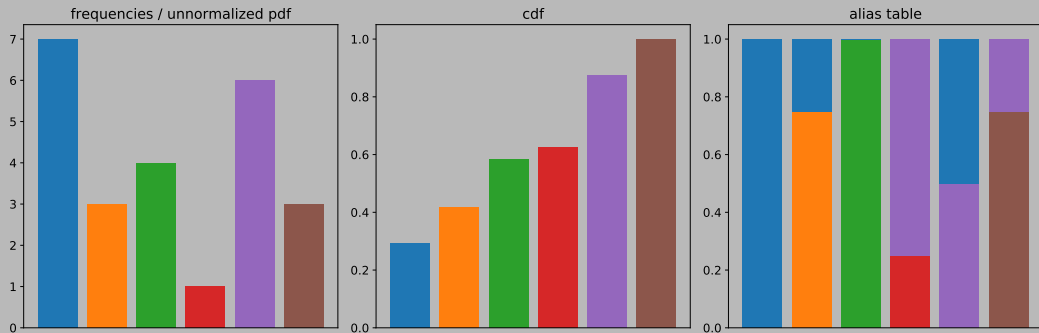


Alias Table Construction



label	A	B	C	D	E	F
split	1	0.75	1	0.25	0.5	0.75
alias		A		E	A	E

Alias Table Construction



Alias Table Interface

label	A	B	C	D	E	F
split	1	0.75	1	0.25	0.5	0.75
alias		A		E	A	E

```
struct Alias
{
    float t;    // split
    int i;      // alias
};

struct AliasTable
{
    int n;
    std::vector<Alias> table;

    AliasTable(const std::vector<float>& values);

    int sample(const float u);
};
```

Alias Table Interface

```
AliasTable::AliasTable(const std::vector<float>& values) : n(values.size()), table(n)
{
    const float sum = std::accumulate( values.begin(), values.end(), 0.f );
    const float avg = sum / n;

    std::vector<int> partition(n);
    int lid = 0, hid = n-1;
    for(int i = 0; i < n; ++i)
    {
        table[i].t = values[i]/avg;
        if( table[i].t <= 1 )
            partition[lid++] = i;
        else
            partition[hid--] = i;
    }

    lid = 0, hid = n-1;
    int tlid = partition[lid], thid = partition[hid], nthid;

    // construct alias table
    while(lid < hid)
    {
        if( table[thid].t > 1 )
        {
            table[tlid].i = thid;
            table[thid].t -= (1 - table[tlid].t);
            tlid = partition[++lid];
        }
        else
        {
            nthid = partition[--hid];
            table[thid].i = nthid;
            table[nthid].t += (1 - table[thid].t);
            thid = nthid;
        }
    }
    // last item should have 100% chance to be picked
    table[thid] = {1, std::numeric_limits<int>::max()};
}
```

- less than 40 loc construction
- 1 temporary array of int
- 3 loops in $O(n)$

Alias Table Interface

```
AliasTable::AliasTable(const std::vector<float>& values) : n(values.size()), table(n)
{
    const float sum = std::accumulate( values.begin(), values.end(), 0.f );
    const float avg = sum / n;

    std::vector<int> partition(n);
    int lid = 0, hid = n-1;
    for(int i = 0; i < n; ++i)
    {
        table[i].t = values[i]/avg;
        if( table[i].t <= 1 )
            partition[lid++] = i;
        else
            partition[hid--] = i;
    }

    lid = 0, hid = n-1;
    int tlid = partition[lid], thid = partition[hid], nthid;

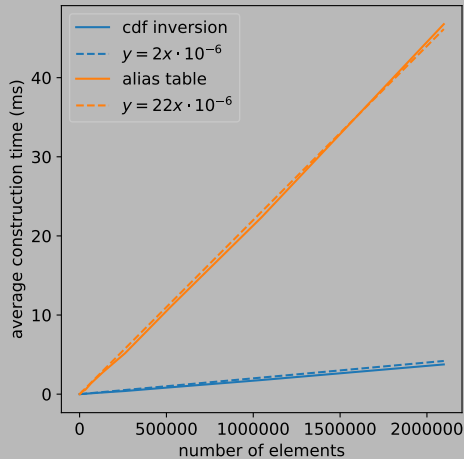
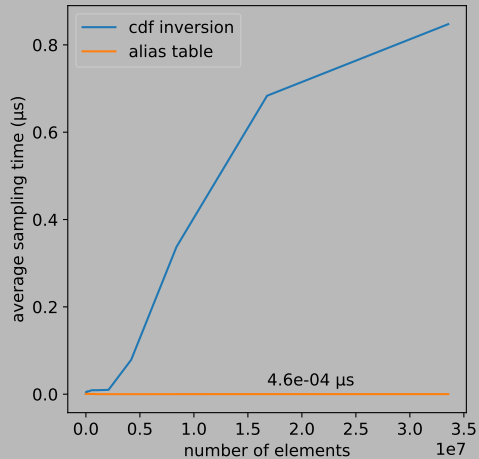
    // construct alias table
    while(lid < hid)
    {
        if( table[thid].t > 1 )
        {
            table[tlid].i = thid;
            table[thid].t -= (1 - table[tlid].t);
            tlid = partition[++lid];
        }
        else
        {
            nthid = partition[--hid];
            table[thid].i = nthid;
            table[nthid].t += (1 - table[thid].t);
            thid = nthid;
        }
    }
    // last item should have 100% chance to be picked
    table[thid] = {1, std::numeric_limits<int>::max()};
}
```

- less than 40 loc construction
- 1 temporary array of int
- 3 loops in $O(n)$

```
int AliasTable::sample(const float u)
{
    const int id = n * u;
    const float u2 = n * u - id;
    return u2 < table[id].t ? id : table[id].i;
}
```

- sampling in $O(1)$ - 3 loc

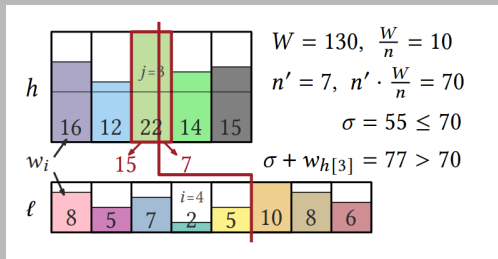
Performance



2-steps Parallel Construction

- divide and conquer approach
- split the array into p subproblems
- a subproblem is determined by finding triplets (i, j, s) such that:

$$\sigma = \sum_{x \leq i} w_{l[x]} + \sum_{x \leq j} w_{h[x]} \leq p \cdot \frac{W}{n} \quad \text{and} \quad w_{h[j+1]} > p \cdot \frac{W}{n} - \sigma = s$$



[Hübschle et al. 2019]

Alias Table Sampling: takeaway

- A structure to sample discrete distributions in constant time
- Favorably compares to CDF inversion for large amounts of samples (e.g. rendering)
- Parallel construction available CPU / GPU