Genome-540 class 16

Chengxiang Qiu

HW7: D-segments Revisited

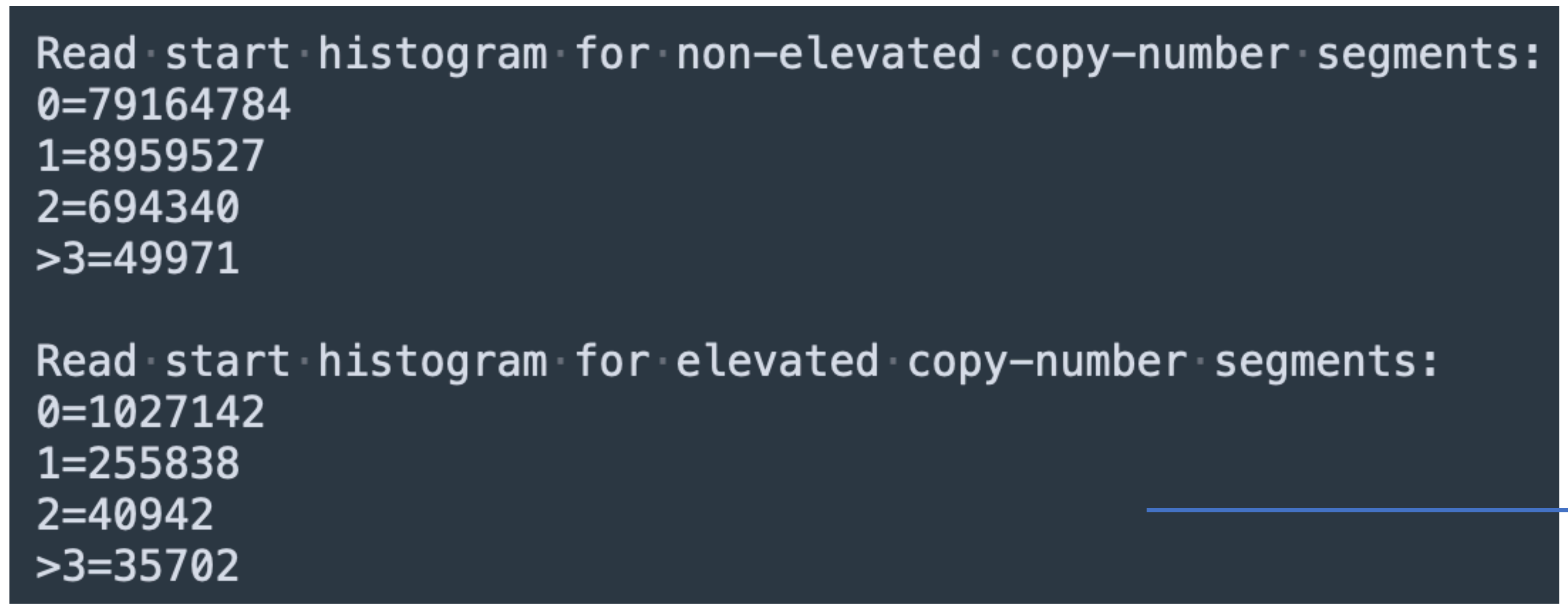
HW7: D-segments Revisited

- **Same input data** as for HW6 (file of read-start counts for chromosome 16)
- **Computing a new scoring scheme** for the read-start bins (0, 1, 2, and >=3)
- **S = -D = 5**

HW7: D-segments Revisited

There are 8,422,401 sites corresponding to sites with 'N' in the reference genome and read alignments cannot start at an 'N'.

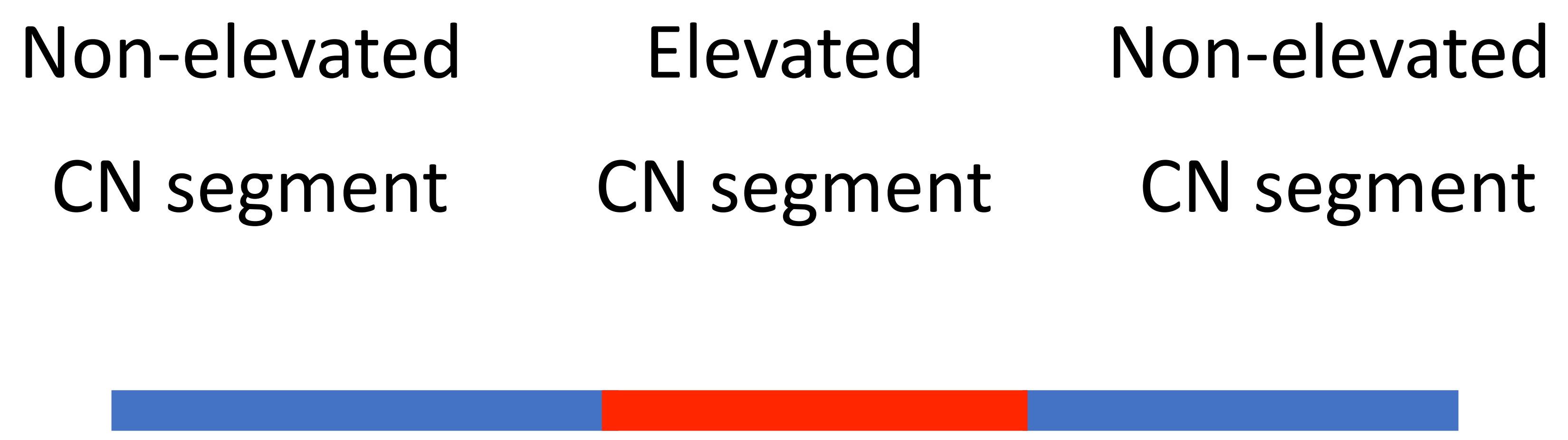
Output of HW6



```
Read start histogram for non-elevated copy-number segments:
0=79164784
1=8959527
2=694340
>3=49971

Read start histogram for elevated copy-number segments:
0=1027142
1=255838
2=40942
>3=35702
```

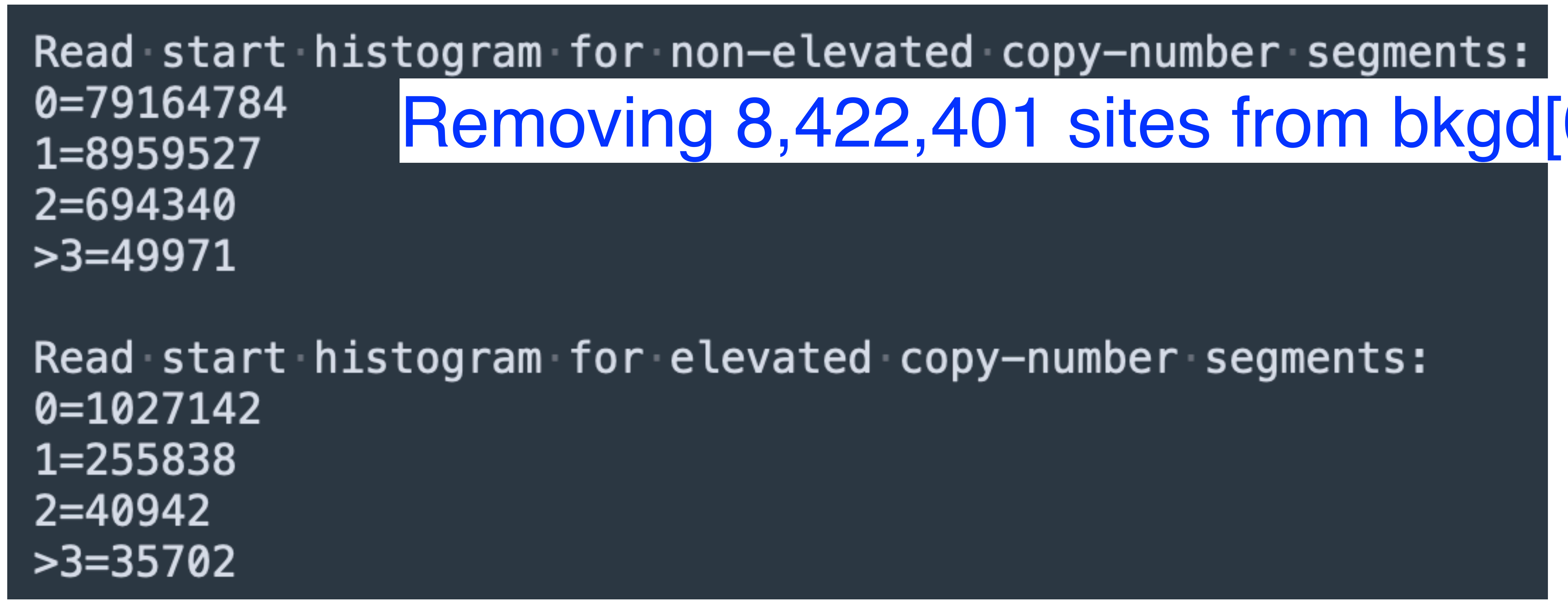
Target

Background



HW7: D-segments Revisited

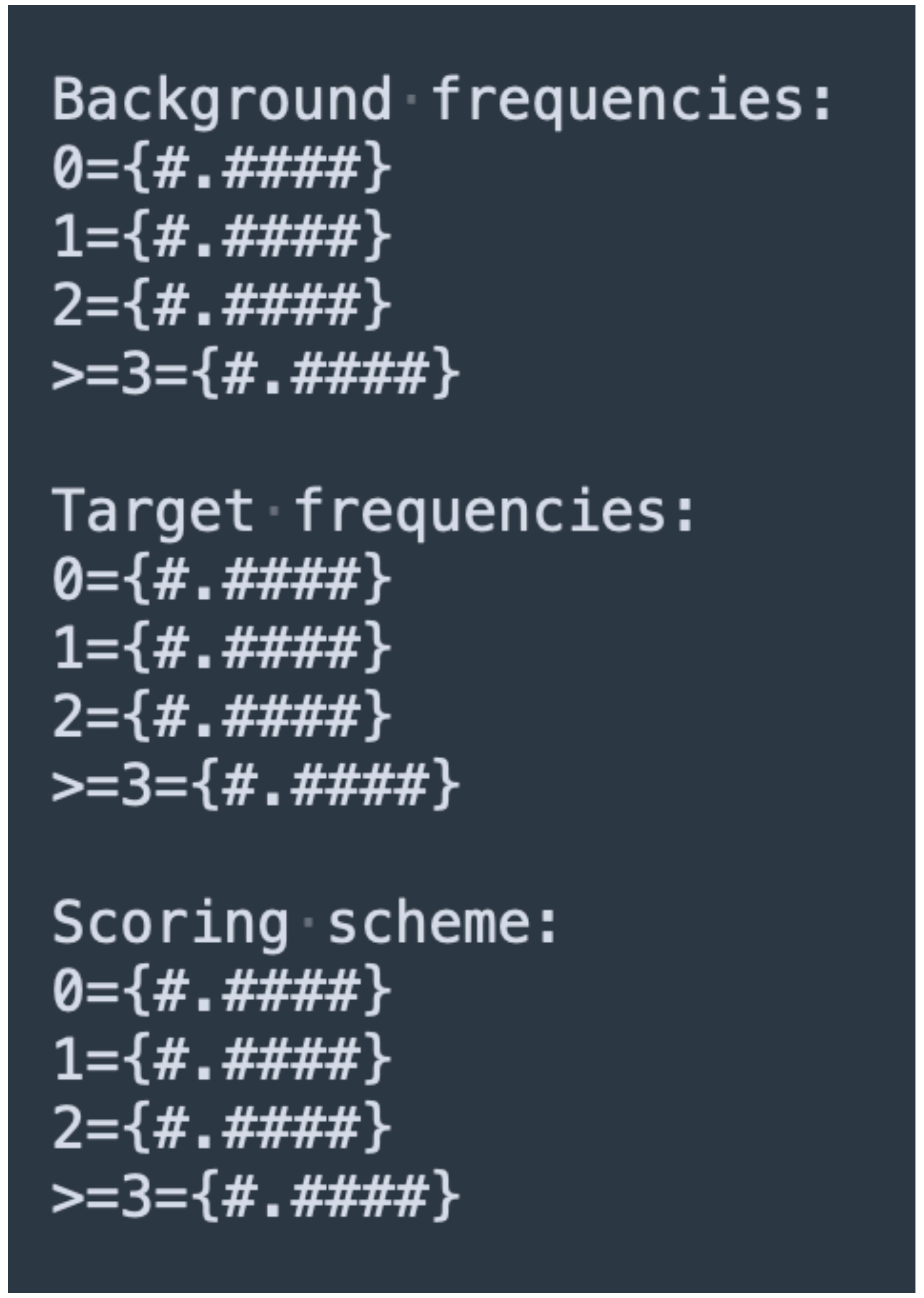
1. Create a scoring scheme (for each count value 0, 1, 2, 3) based on the background and target frequencies, using LLRs with base 2 logarithms



```
Read start histogram for non-elevated copy-number segments:
0=79164784
1=8959527
2=694340
>3=49971

Read start histogram for elevated copy-number segments:
0=1027142
1=255838
2=40942
>3=35702
```

Removing 8,422,401 sites from bkgd[0]

log2(freq_target/freq_background)

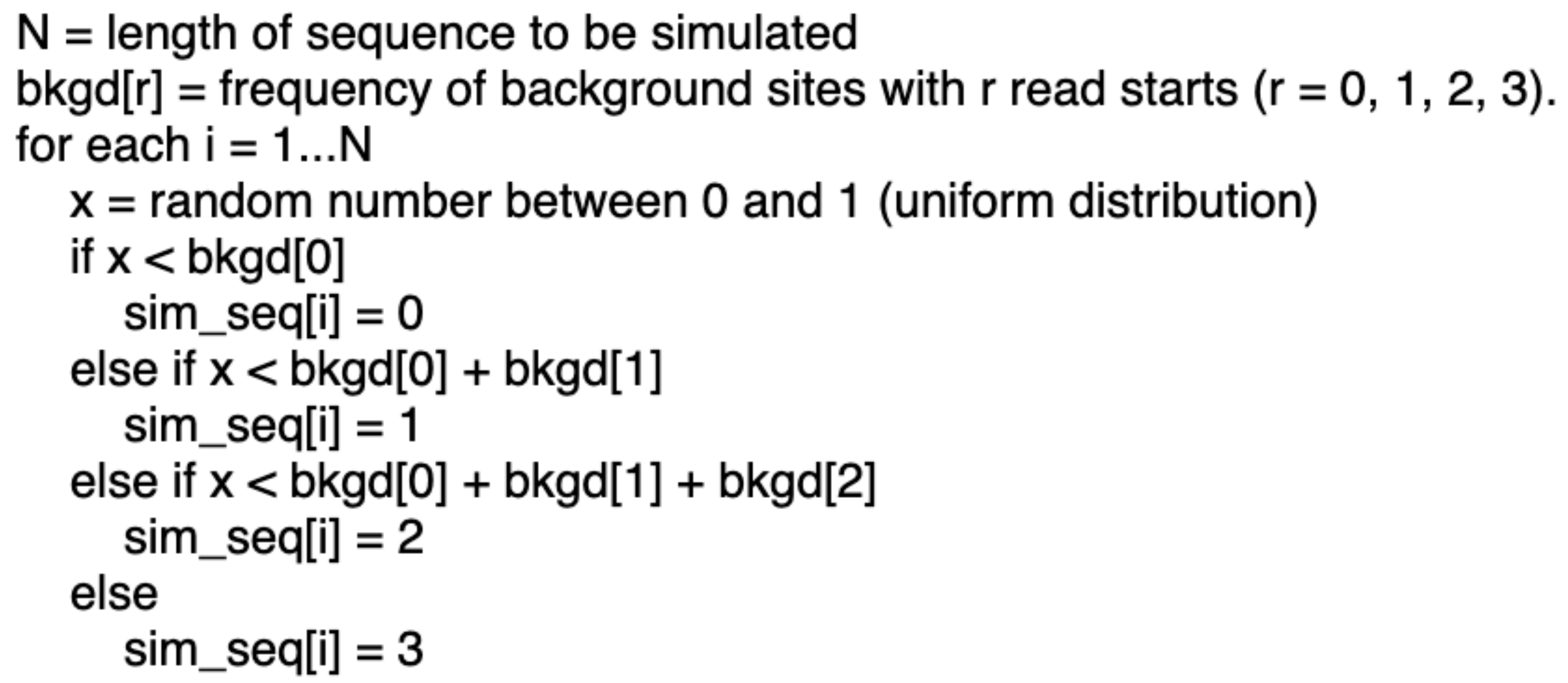


```
Background frequencies:
0={#.####}
1={#.####}
2={#.####}
>=3={#.####}

Target frequencies:
0={#.####}
1={#.####}
2={#.####}
>=3={#.####}

Scoring scheme:
0={#.####}
1={#.####}
2={#.####}
>=3={#.####}
```

HW7: D-segments Revisited

2. Write a program that uses the background frequencies above to simulate a sequence of read start counts. The length of this sequence should be the total length of the chromosome used in HW6 minus the number of N's (as given above).

Background



```
N = length of sequence to be simulated
bkgd[r] = frequency of background sites with r read starts (r = 0, 1, 2, 3).
for each i = 1...N
    x = random number between 0 and 1 (uniform distribution)
    if x < bkgd[0]
        sim_seq[i] = 0
    else if x < bkgd[0] + bkgd[1]
        sim_seq[i] = 1
    else if x < bkgd[0] + bkgd[1] + bkgd[2]
        sim_seq[i] = 2
    else
        sim_seq[i] = 3
```

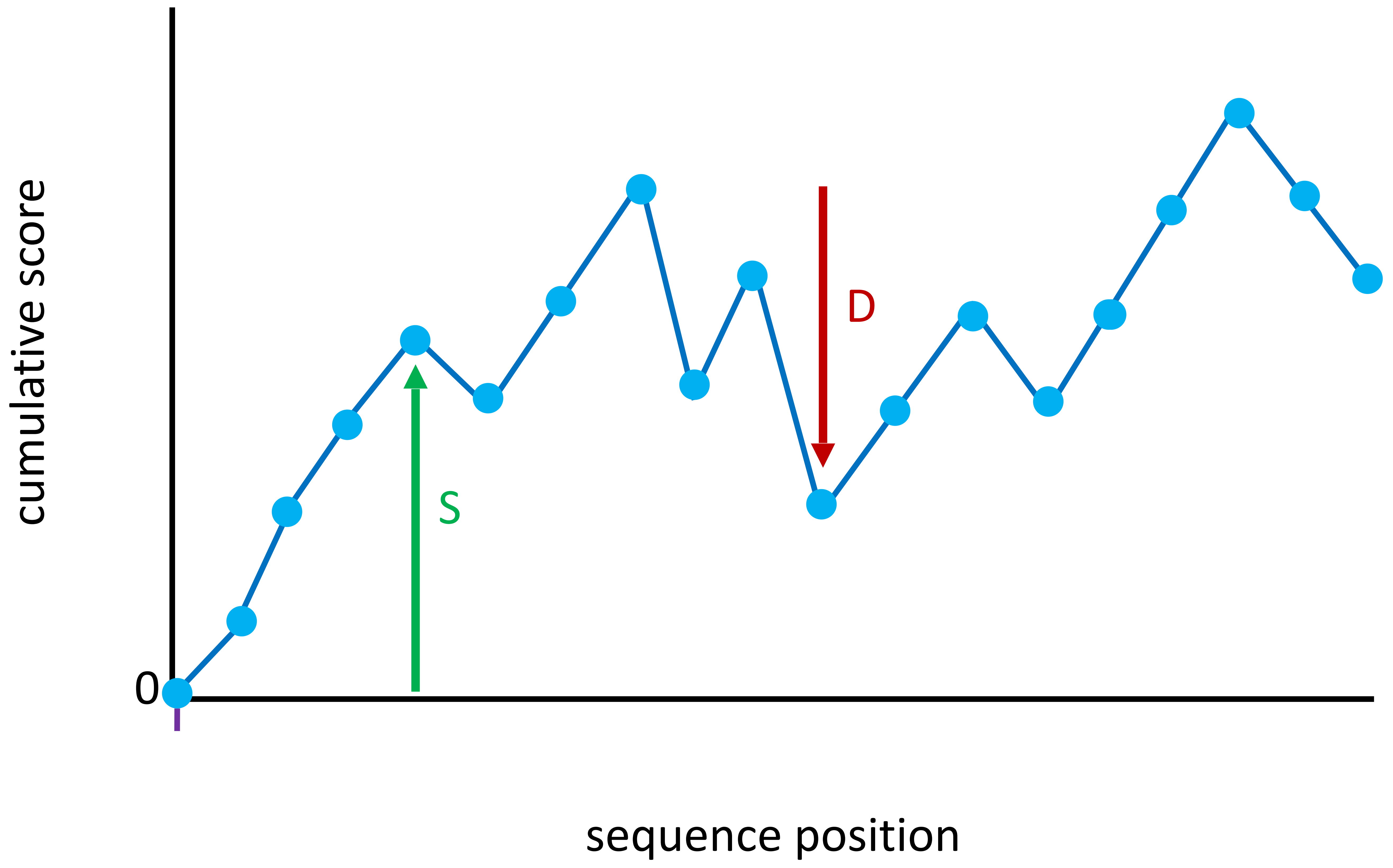
HW7: D-segments Revisited

3. Run your maximal D-segment algorithm on the simulated count sequence with S = -D = 5 and the above scoring scheme. Report a list of pairs, giving for each integer score s = 5, ... 30 the number N_seg(s) of D-segments with score >= s.

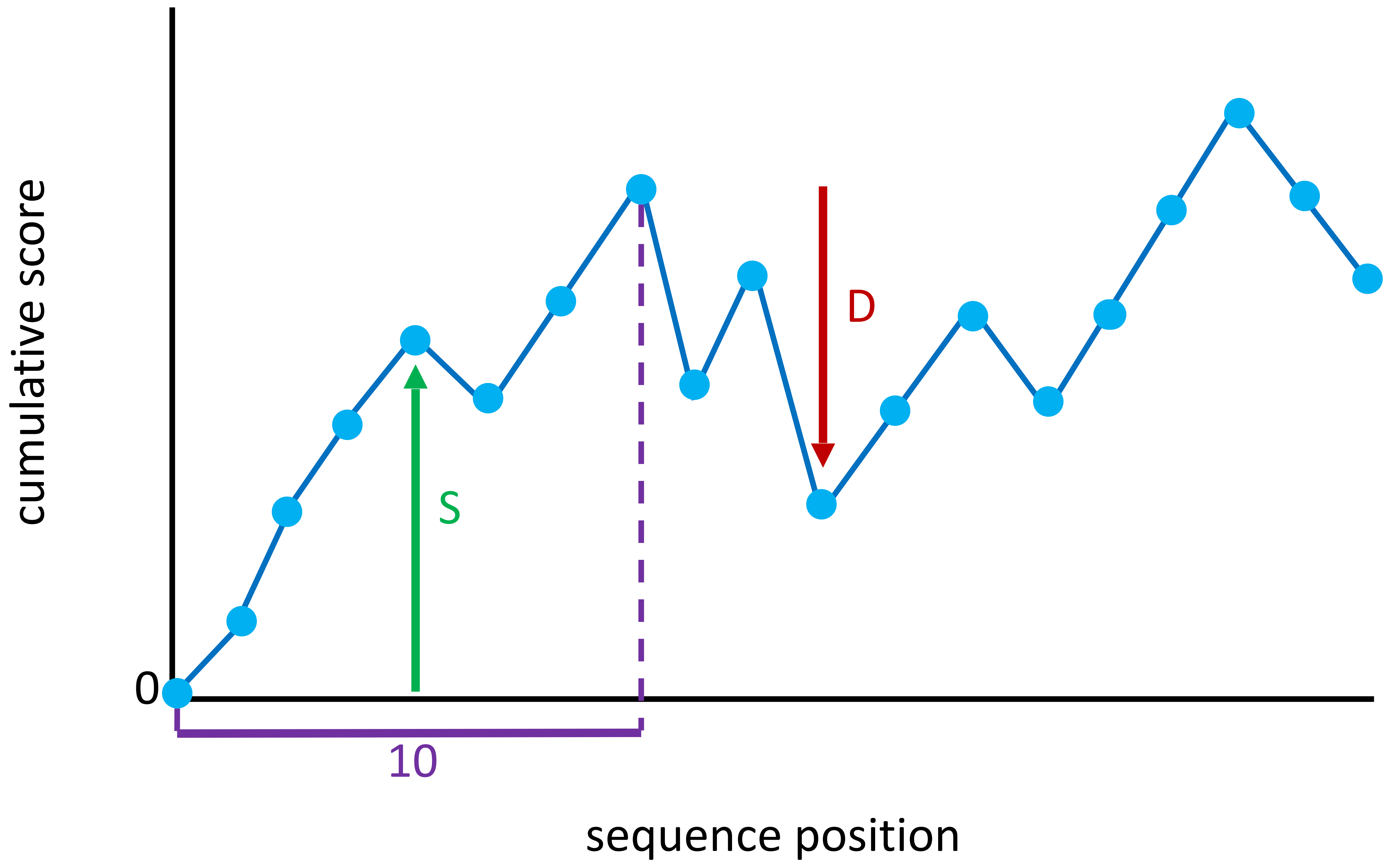
4. Run your maximal D-segment algorithm on the 'real data' sequence of read starts used in assignment 6 with the above S and D values, scoring scheme, and list output.

We care about

`{# of segments with score >= S}`



```
Simulated data:
5  0
6  0
7  0
8  0
9  0
10  0
```



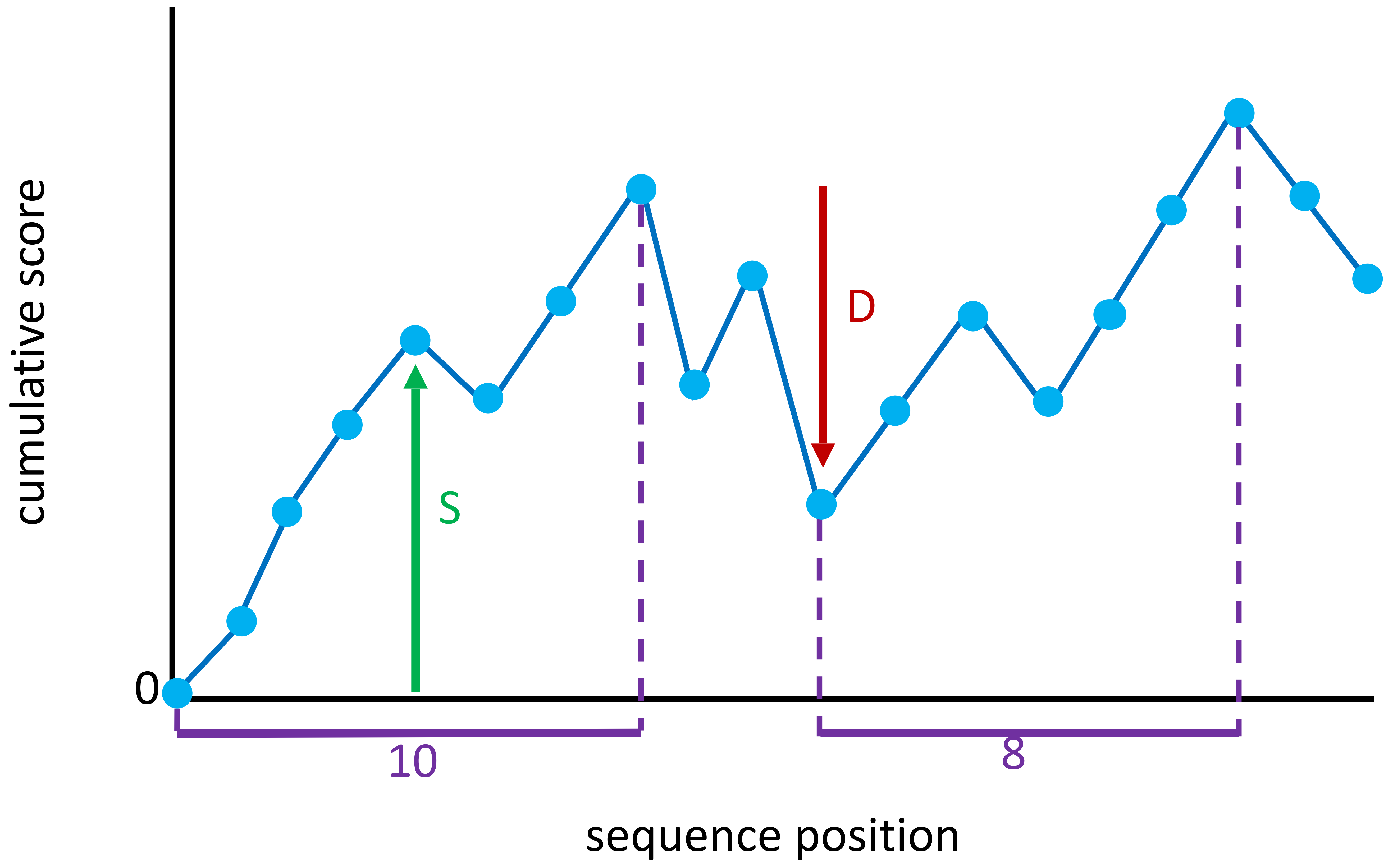
We care about

`{# of segments with score >= S}`

```
Simulated data:
5 1
6 1
7 1
8 1
9 1
10 1
```

We care about

`{# of segments with score >= S}`



```
Simulated data:
5  2
6  2
7  2
8  2
9  1
10  1
```

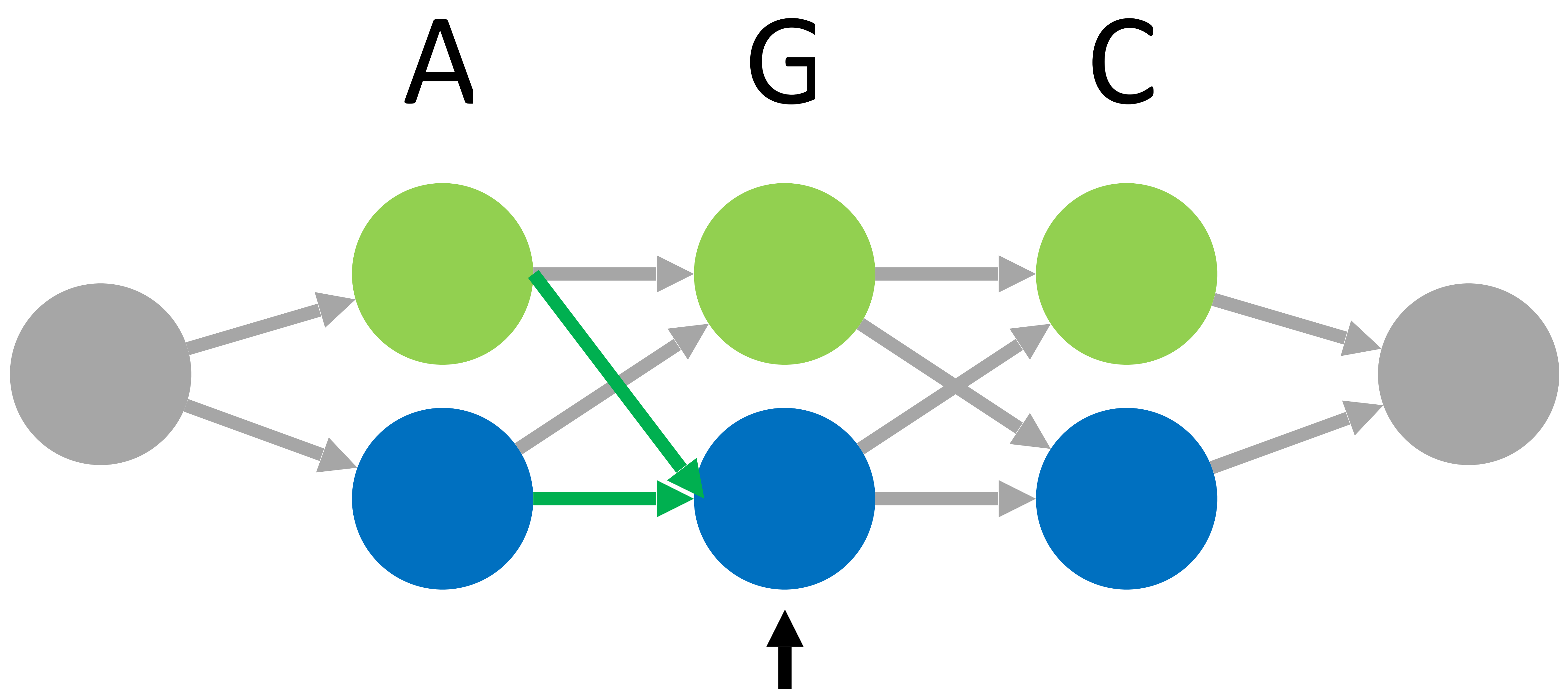
HW7: D-segments Revisited

- Output:
  - Background/Target frequencies, and scoring matrix
  - Two lists of pairs, one for the original ‘real’ data and another for the simulated data. Each row should contain:
    - S-value
    - Number of D-segments found
  - A list of ratios based on the simulated data:
    - Label each row N_seg(S_i)/N_seg(S_{i+1})
    - Ratio of #D-seg(S_i)/#D-seg(S_{i+1}) rounded to 2 dec.
    - If there is a 0 in the denominator of your ratio, print -1
  - Brief written answers to the questions posed in the assignment text

```
Real data:
5 {# of segments with score >= 5}
6 {# of segments with score >= 6}
7 {# of segments with score >= 7}
.
.

Simulated data:
5 {# of segments with score >= 5}
6 {# of segments with score >= 6}
7 {# of segments with score >= 7}
.
.

Ratios of simulated data:
N_seg(5)/N_seg(6){ratio}
N_seg(6)/N_seg(7) {ratio}
N_seg(7)/N_seg(8) {ratio}
.
.
```

HW7: Questions?

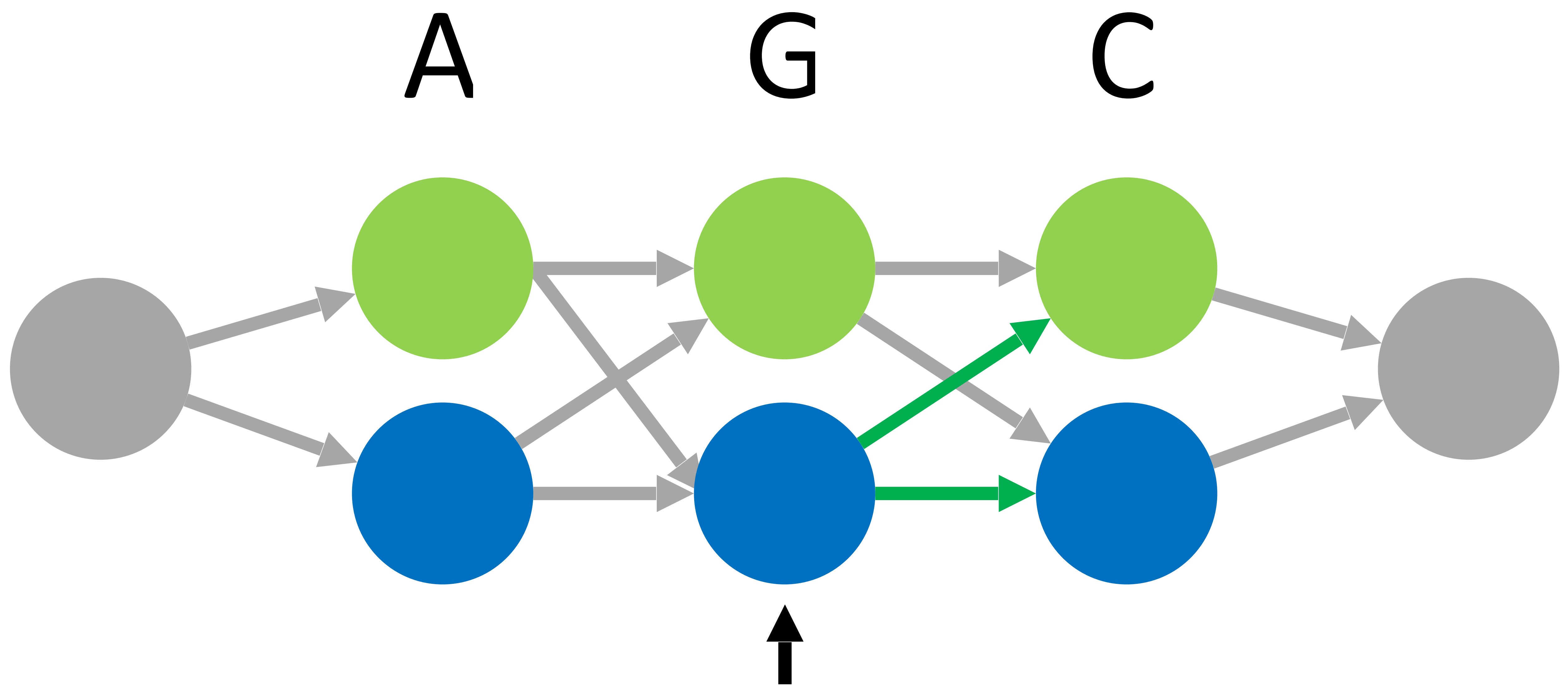
Forward-backward algorithm



For each node:

- Forward: store the sum of probabilities of paths ending at position *t* state *i*

Forward-backward algorithm



For each node:

- Forward: store the sum of probabilities of paths ending at position *t* state *i*
- Backward: store the sum of probabilities of paths starting at position *t* state *i*

Forward Algorithm

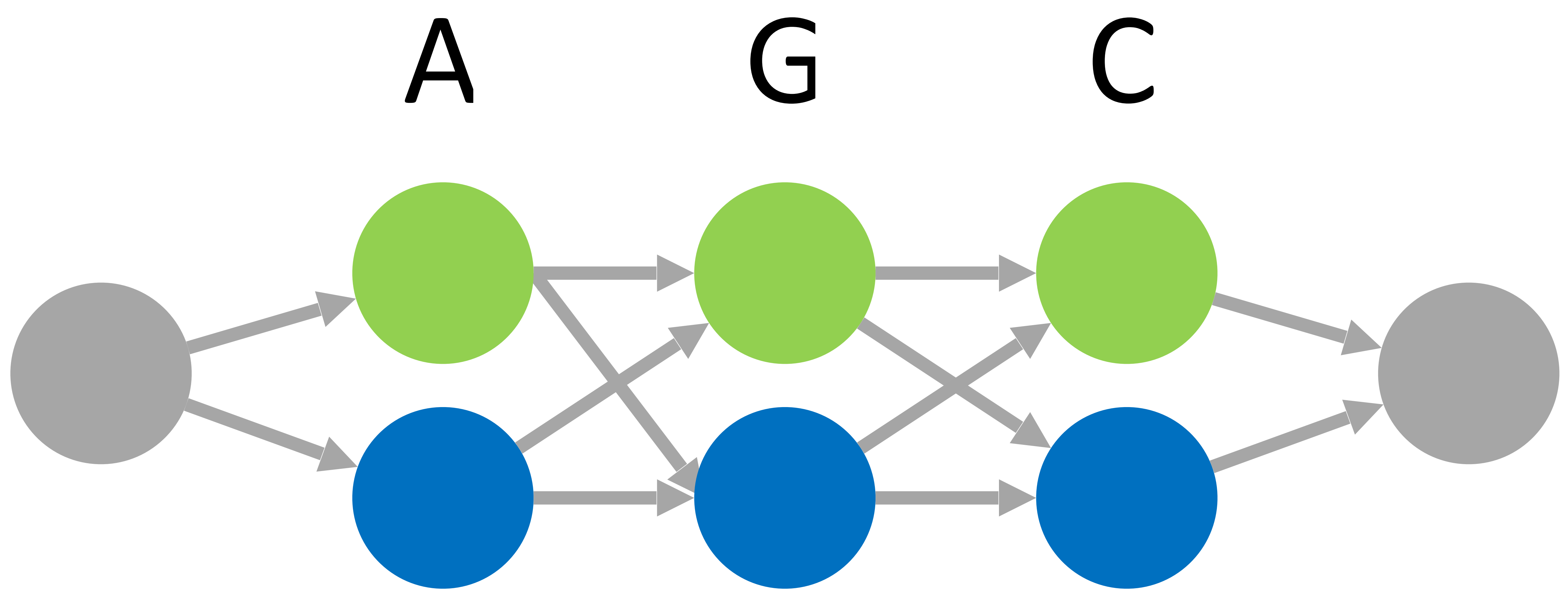
1. Initialization:

$$\alpha_1(i) = \pi_i b_i(O_1), \qquad 1 \leq i \leq N$$

2. Induction:

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i) a_{ij} \right] b_j(O_{t+1}), \qquad 1 \leq t \leq T-1, 1 \leq j \leq N.$$

Build a dynamic programming table for these calculations



	A	G	C
State 1			
State 2			

Backward Algorithm

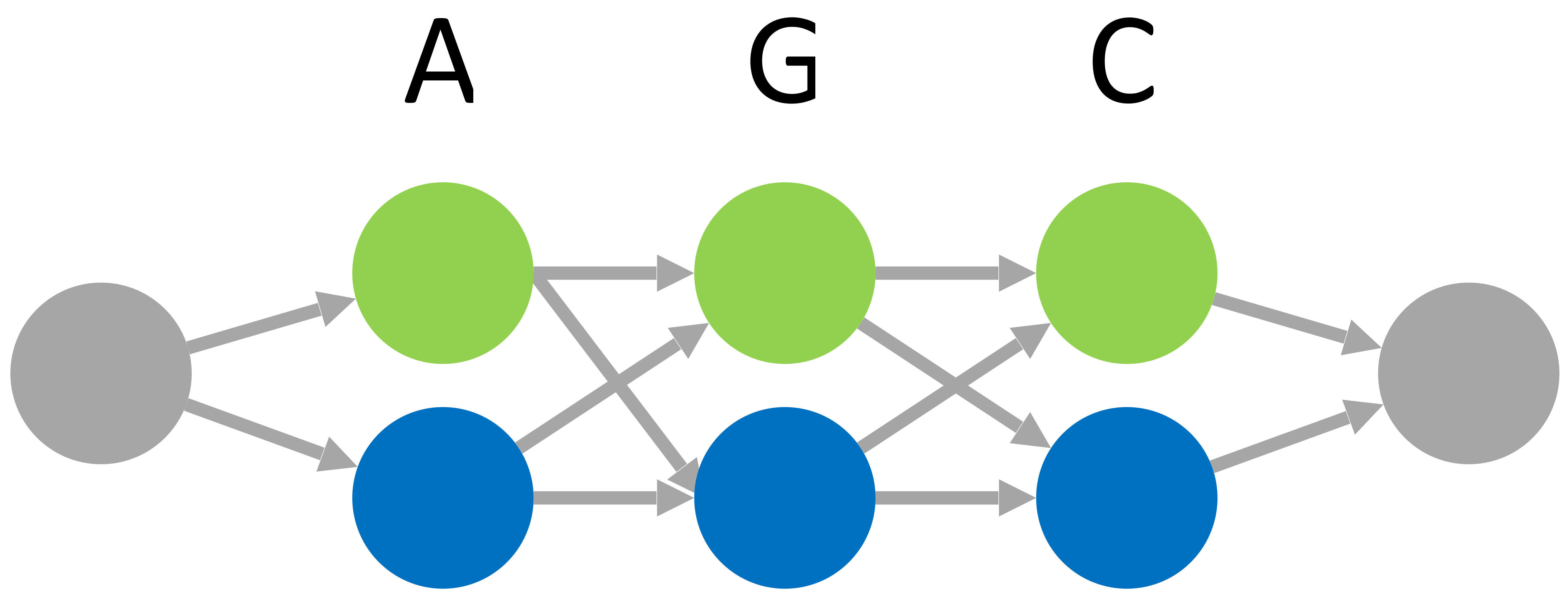
1. Initialization:

$$\beta_T(i) = 1, \qquad 1 \le i \le N$$

2. Induction:

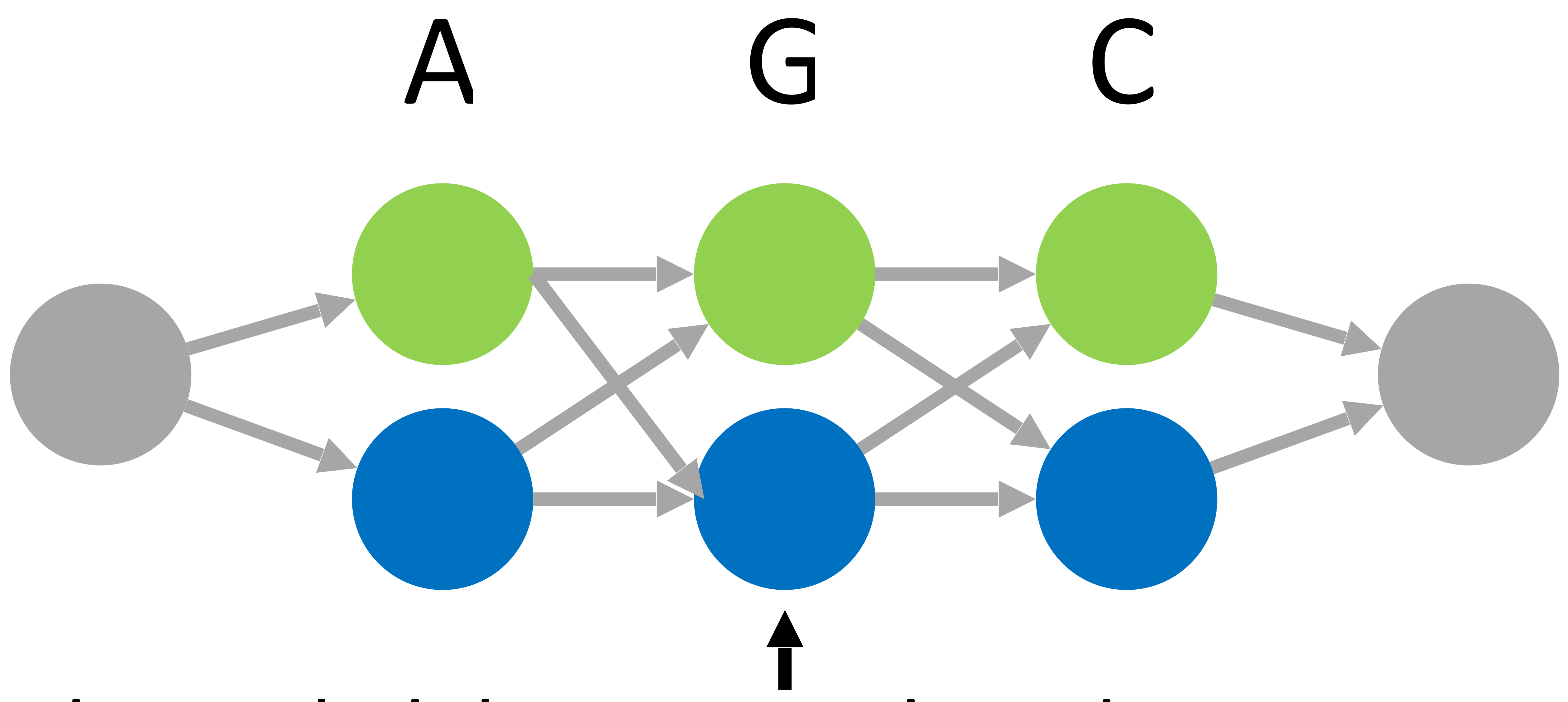
$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(O_{t+1}) \beta_{t+1}(j), \qquad 1 \le t \le T-1, 1 \le j \le N.$$

Build a dynamic programming table for these calculations



	A	G	C
State 1			
State 2			

Think about how to update parameters (A,B,Pi)

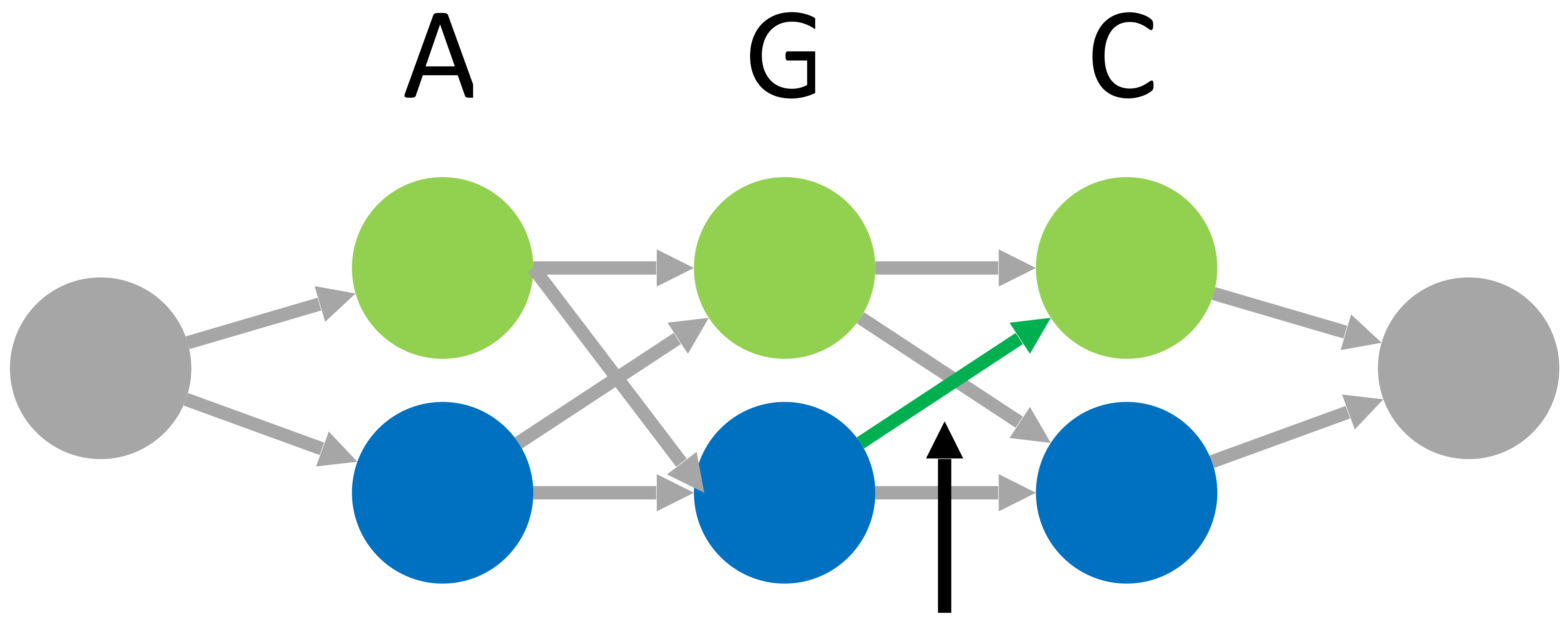


Consider the probabilities at each node:

- Figure out the probability of being in state *i* at position *t*

$$\gamma_t(i) = \frac{\alpha_t(i)\,\beta_t(i)}{P(O|\lambda)} = \frac{\alpha_t(i)\,\beta_t(i)}{\sum_{i=1}^{N} \alpha_t(i)\,\beta_t(i)}$$

Think about how to update parameters (A,B,Pi)



Consider the probabilities at each edge:

- Figure out the probability of going from state *i* to state t from position *j* to position *t*+1

$$\xi_t(i,j) = \frac{\alpha_t(i)\, a_{ij} b_j(O_{t+1})\, \beta_{t+1}(j)}{P(O|\lambda)}$$

$$= \frac{\alpha_t(i)\, a_{ij} b_j(O_{t+1})\, \beta_{t+1}(j)}{\sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_t(i)\, a_{ij} b_j(O_{t+1})\, \beta_{t+1}(j)}$$

Think about how to update parameters (A,B,Pi)

$$\bar{\pi}_i = \text{expected frequency (number of times) in state } S_i \text{ at time } (t = 1) = \gamma_1(i)$$

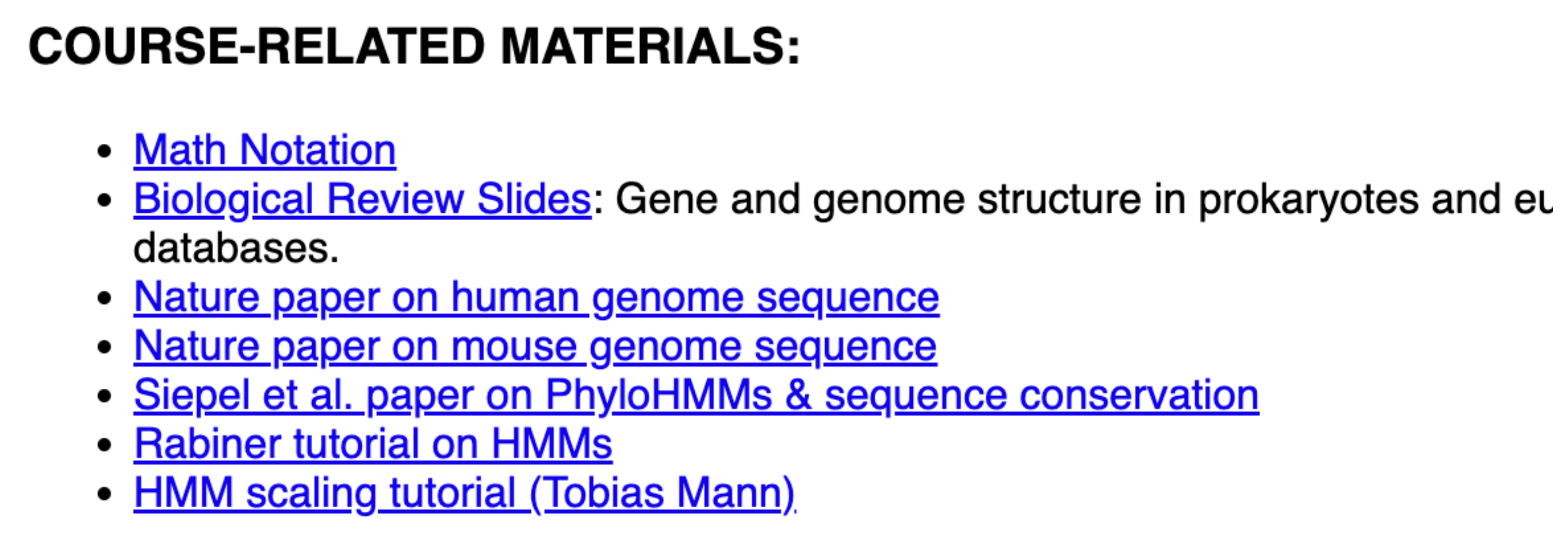
$$\bar{a}_{ij} = \frac{\text{expected number of transitions from state } S_i \text{ to state } S_j}{\text{expected number of transitions from state } S_i}$$

$$= \frac{\sum_{t=1}^{T-1} \xi_t(i, j)}{\sum_{t=1}^{T-1} \gamma_t(i)}$$

$$\bar{b}_j(k) = \frac{\text{expected number of times in state } j \text{ and observing symbol } v_k}{\text{expected number of times in state } j}$$

$$= \frac{\sum_{\substack{t=1 \\ \text{s.t. } O_t = v_k}}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)}.$$

Underflow - this is very important



COURSE-RELATED MATERIALS:

- Math Notation
- Biological Review Slides: Gene and genome structure in prokaryotes and eu databases.
- Nature paper on human genome sequence
- Nature paper on mouse genome sequence
- Siepel et al. paper on PhyloHMMs & sequence conservation
- Rabiner tutorial on HMMs
- HMM scaling tutorial (Tobias Mann)

- This happens when numbers are too small to be stored in a variable

Solutions:

- Scale weights to be close to 1 (affects all paths by same constant factor – which can be multiplied back later)
- Use log weights, so can add instead of multiplying
- Ex: Instead of 0.0001 * 0.0002, you can do:

  log(0.0001) + log(0.0002)

What about when you need to sum probabilities in logspace? See this blogpost for a solution or Tobias Mann

https://gasstationwithoutpumps.wordpress.com/2014/05/06/sum-of-probabilities-in-log-prob-space/

Scale

Forward

- Initialization

$$\begin{aligned}\ddot{\alpha}_1(i) &= \alpha_1(i)\\ c_1 &= \frac{1}{\sum_{i=1}^N \ddot{\alpha}_1(i)}\\ \hat{\alpha}_1(i) &= c_1\ddot{\alpha}_1(i)\end{aligned}$$

- Induction

$$\begin{aligned}\ddot{\alpha}_t(i) &= \sum_{j=1}^N \hat{\alpha}_{t-1}(j)a_{ji}b_i(O_t)\\ c_t &= \frac{1}{\sum_{i=1}^N \ddot{\alpha}_t(i)}\\ \hat{\alpha}_t(i) &= c_t\ddot{\alpha}_t(i)\end{aligned}$$

	A	C	G
State 1			
State 2			
	c1	c2	c3

$$\hat{\alpha}_t(i) = \left(\prod_{\tau=1}^{t} c_\tau\right)\alpha_t(i).$$

$$\mathbf{C}_t = \prod_{\tau=1}^{t} c_\tau$$

$$\boxed{\log[P(O|\lambda)] = -\sum_{t=1}^{T}\log c_t.}$$

Scale

Backward

- Initialization

$$\begin{aligned}\ddot{\beta}_T(i) &= 1 \\ \hat{\beta}_T(i) &= c_T \ddot{\beta}_T(i)\end{aligned}$$

$$\hat{\beta}_t(i) = \left(\prod_{s=t}^{T} c_s\right) \beta_t(i) = \mathbf{D}_t \beta_t(i),$$

- Induction

$$\begin{aligned}\ddot{\beta}_t(i) &= \sum_{j=1}^{N} a_{ij} b_j(O_{t+1}) \hat{\beta}_{t+1}(j) \\ \hat{\beta}_t(i) &= c_t \ddot{\beta}_t(i)\end{aligned}$$

Scale

$$\begin{aligned}\bar{b}_j(k) &= \frac{\sum_{t=1,O_t=v_k}^{T}\gamma_t(j)}{\sum_{t=1}^{T}\gamma_t(j)} \\ &= \frac{\sum_{t=1,O_t=v_k}^{T}\hat{\alpha}_t(j)\cdot\hat{\beta}_t(j)/c_t}{\sum_{t=1}^{T}\hat{\alpha}_t(j)\cdot\hat{\beta}_t(j)/c_t}.\end{aligned}$$

$$\begin{aligned}\bar{a}_{ij} &= \frac{\sum_{t=1}^{T-1}\xi_t(i,j)}{\sum_{t=1}^{T-1}\gamma_t(i)} \\ &= \frac{\sum_{t=1}^{T-1}\alpha_t(i)a_{ij}b_j(O_{t+1})\beta_{t+1}(j)}{\sum_{t=1}^{T-1}\alpha_t(i)\beta_t(i)} \\ &= \frac{\sum_{t=1}^{T-1}\hat{\alpha}_t(i)/\mathbf{C}_t\cdot a_{ij}b_j(O_{t+1})\cdot\hat{\beta}_{t+1}(j)/\mathbf{D}_{t+1}}{\sum_{t=1}^{T-1}\hat{\alpha}_t(i)/\mathbf{C}_t\cdot\hat{\beta}_t(i)/\mathbf{D}_t} \\ &= \frac{\left(\sum_{t=1}^{T-1}\hat{\alpha}_t(i)\cdot a_{ij}b_j(O_{t+1})\cdot\hat{\beta}_{t+1}(j)\right)/\mathbf{C}_T}{\left(\sum_{t=1}^{T-1}\hat{\alpha}_t(i)\cdot\hat{\beta}_t(i)/c_t\right)/\mathbf{C}_T} \\ &= \frac{\sum_{t=1}^{T-1}\hat{\alpha}_t(i)\cdot a_{ij}b_j(O_{t+1})\cdot\hat{\beta}_{t+1}(j)}{\sum_{t=1}^{T-1}\hat{\alpha}_t(i)\cdot\hat{\beta}_t(i)/c_t}.\end{aligned}$$

initial prob
r1(i) = a1(i) * b1(i) / c1

$$\log[P(O|\lambda)] = -\sum_{t=1}^{T}\log c_t.$$

HW8: detecting G+C-rich regions (Baum-Welch)

- Due 11:59pm Sunday, March 6
- Assignment: use Baum-Welch algorithm to identify G+C-rich regions in a genome sequence
  - Input: FASTA
  - Run Baum-Welch until the increase in sequence log-likelihood is less than 0.1
  - Output:
    - Name and first line of the FASTA file
    - Number of iterations until convergence
    - Final sequence log-likelihood
    - Final probabilities (initial, transition, emission)
      - Scientific notation, four significant digits (i.e., 9.000e-1; see template)

$$\log[P(O|\lambda)]$$

Notes for debugging

1. Try calculating some simple forward and backward probabilities by hand to check your algorithm

2. **The likelihood at each iteration should increase; if it decreases, then you have a bug**

3. Have a print statement in your program to keep track of iterations as your program is running. The assignment will provide an estimate on the number of iterations to converge.