CSE 566 Spring 2023

Hirschberg's Algorithm

Instructor: Mingfu Shao

Space Complexity

- Algorithms that use O(mn) space, $m = |S|$, $n = |T|$
  - Global alignment with unit gap cost
  - Global alignment with affine gap cost
  - Local alignment with unit gap cost
  - Local alignment with affine gap cost
- Not affordable for comparing long sequences:
  - Human 1st chrm (247M) vs mouse 1st chrm (195M)
  - Memory = 48,000TB

Linear Space for Optimal Value

- Consider global alignment with unit gap cost.
- What if we only need to find the optimal cost, i.e., the optimal alignment is not required?

		A	C	C	G	T
A						
C						
A						
T						

Linear Space for Optimal Value

- Maintain two rows, previous row and current row; update current row using the previous row.

```
algorithm DP-opt-cost(S[1..m], T[1..n])
    Initialize the first row -> F1;
    for i = 1 to m
        fill up array F using F1; /*ith row*/
        F1 = F;
    end for
    return F[n];
end algorithm
```

Linear Space for Optimal Alignment

- The previous approach does not work, as the trackback pointers are not stored.
- Hirschberg's algorithm (1975); idea: divide-and-conquer

```
algorithm merge-sort(A[1..n])
    S = merge-sort(A[1..n/2];
    T = merge-sort(A[n/2+1..n]);
    return merge-two-sorted-arrays(S, T);
end algorithm
```

$T(n) = 2T(n/2) + O(n)$

$\Rightarrow T(n) = O(n \log n)$

$P(n) = \max\{P(n/2), P(n/2), \underbrace{O(n)}^{cn}\}$

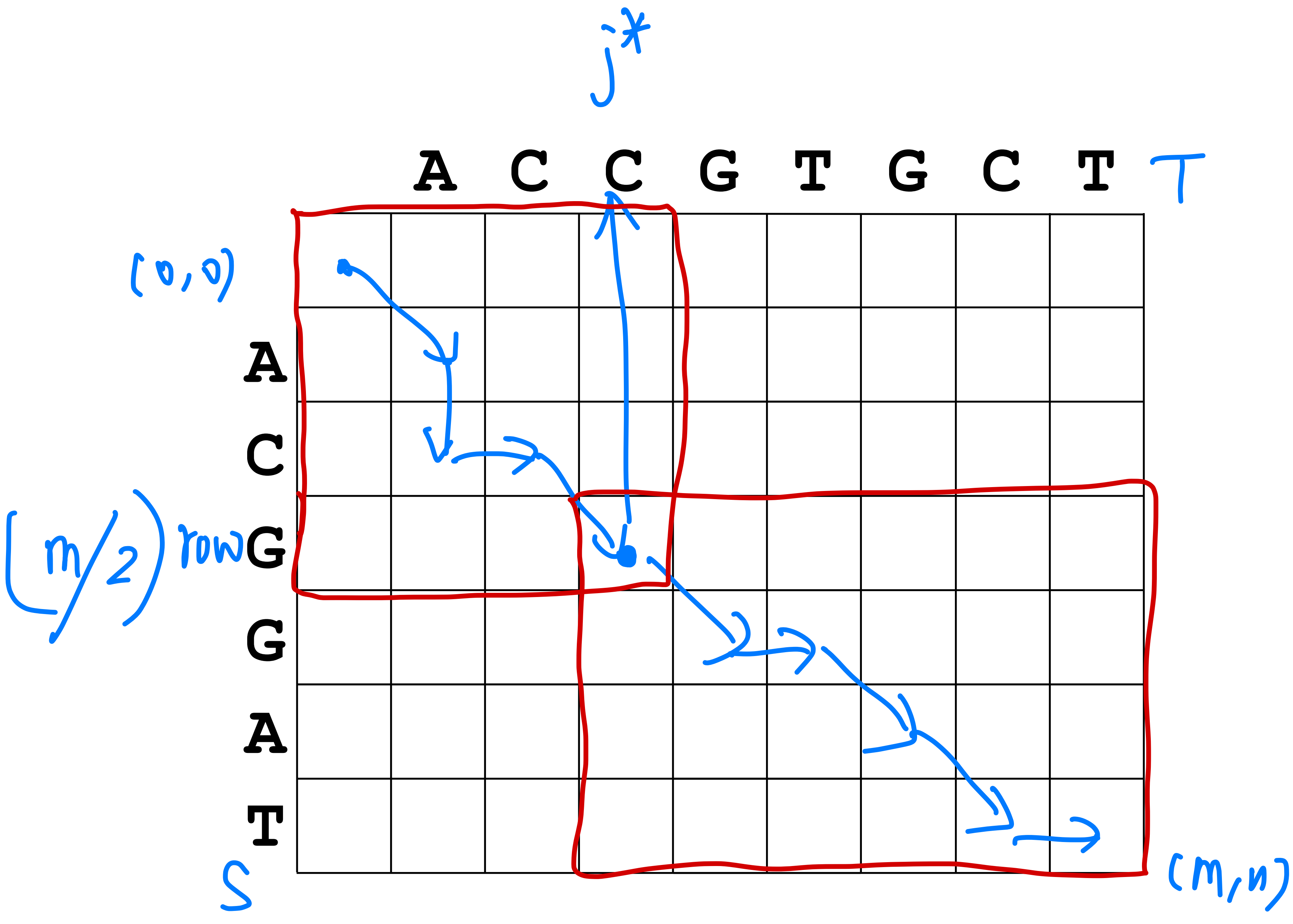
$P(n) \le cn$

$\le \max\{cn/2, cn/2, cn\}$

$= cn$

$\Rightarrow P(n) = O(n)$

Framework of Hirschberg's Algorithm



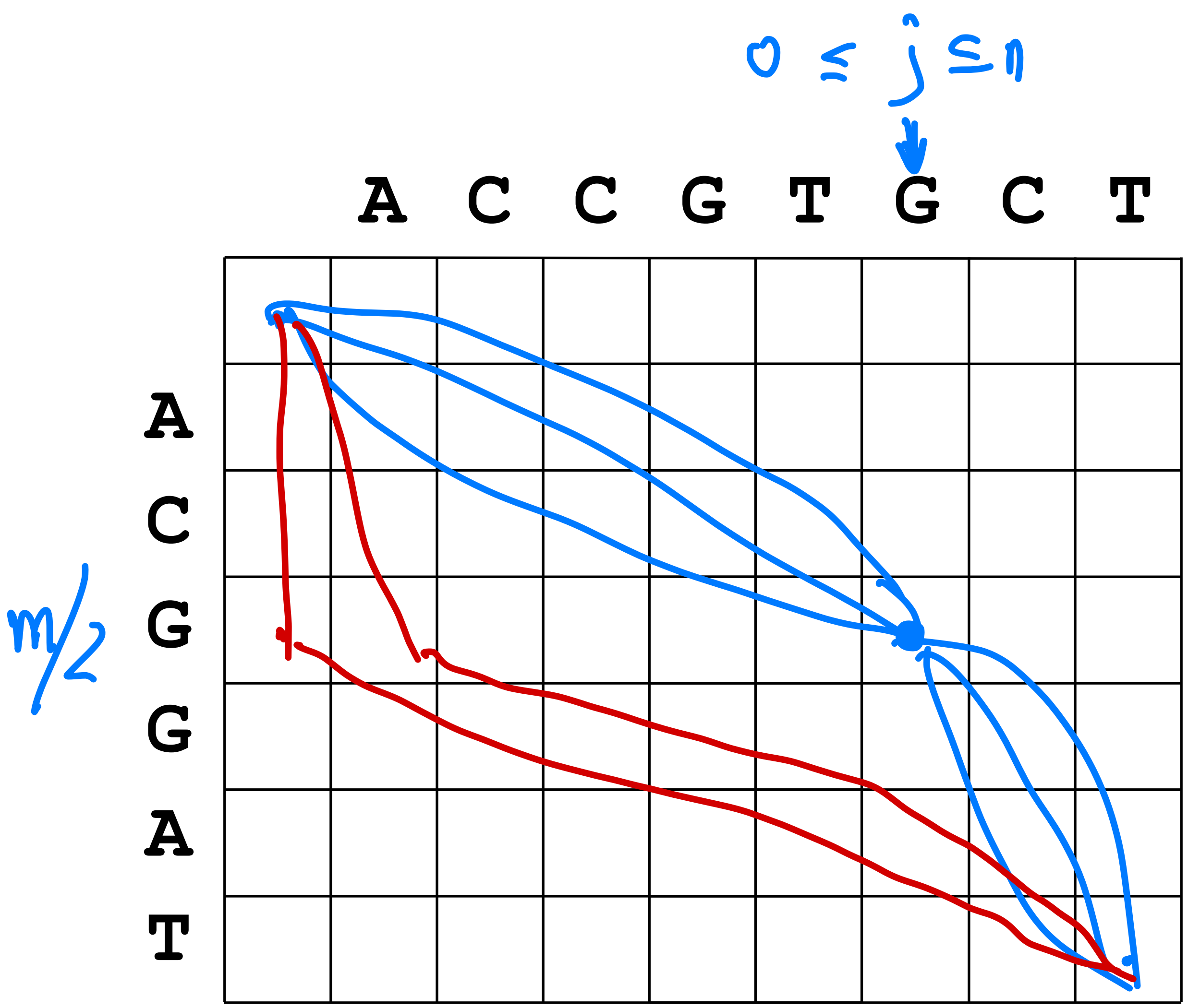
Framework of Hirschberg's Algorithm

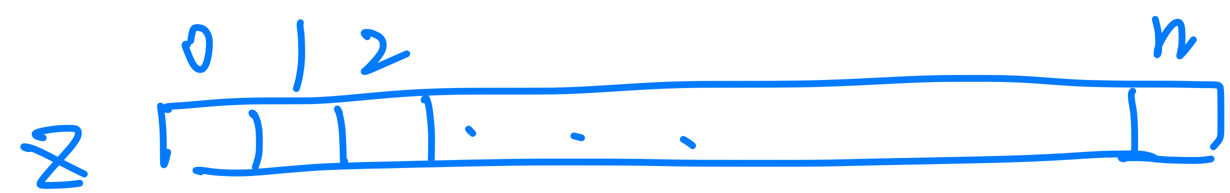
- Define *hirschberg(S, T)* returns the optimal path from (0,0) to (m,n); define that the returned path excludes (0,0) and (m,n).

```
algorithm hirschberg(S[1..m], T[1..n])
    j* = find-middle(S,T);
    P1 = hirschberg(S[1..m/2], T[1..j*];
    P2 = hirschberg(S[m/2..m], T[j*..n];
    return P1+(m/2,j*)+P2;
end algorithm
```

(m/2, j*)

Find-Middle



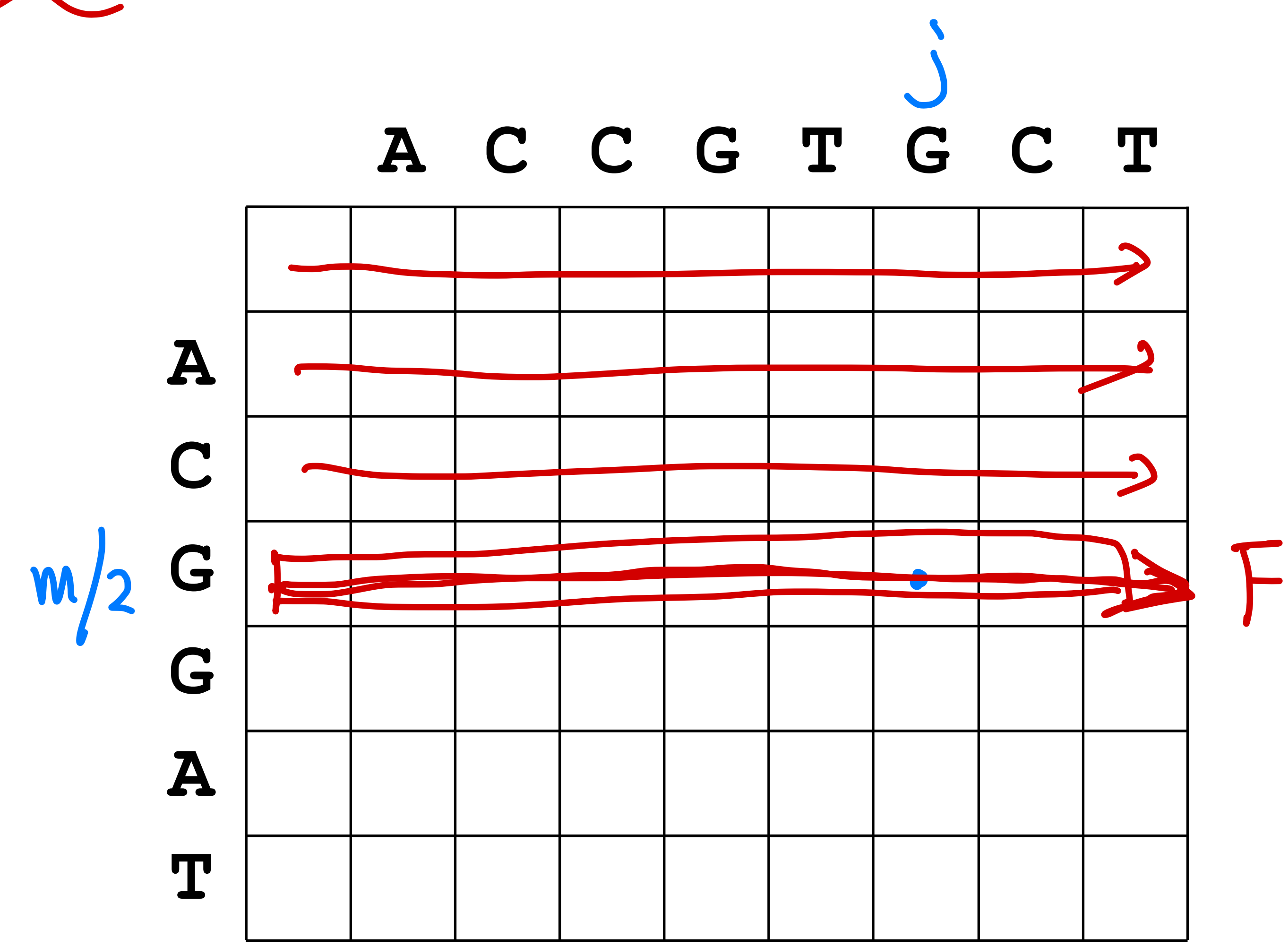


- Define Z(j) as the minimized cost of paths that cross (m/2, j).
- $OPT = \min_{0 \leq j \leq n} Z(j)$
- $j^* = \arg\min_{0 \leq j \leq n} Z(j)$

Calculating Z(j), for all $0 \le j \le n$

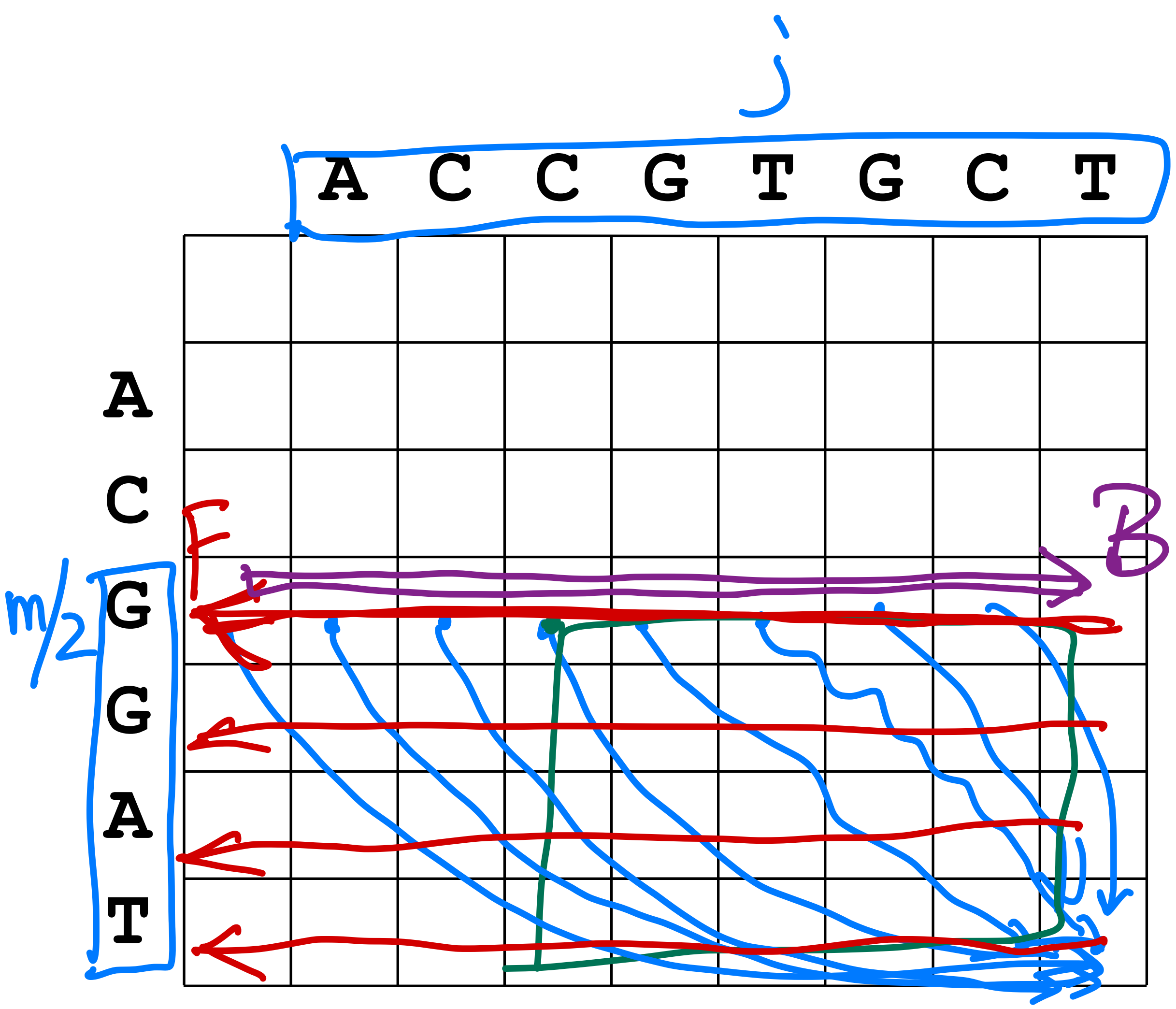
Z(j) = opt-cost from (0,0) to (m/2,j) + opt-cost from (m/2,j) to (m,n)

- Exactly the solution of subproblem OPT(m/2, j)
- Call `DP-opt-cost (S[1..m/2], T[1..n])`, then use the final F array
- F[j] stores the opt-cost from (0,0) to (m/2,j), for any $0 \le j \le n$



Calculating Z(j), for all $1 \leq j \leq n$

Z(j) = opt-cost from (0,0) to (m/2,j) + opt-cost from (m/2,j) to (m,n)



- Call `DP-opt-cost (S[m..m/2], T[n..1])`, then use the resulting F array
- Let B be the reverse of F
- B[j] stores the opt-cost from (m/2,j) to (m,n)

DP-opt-cost (TAGG, TCGTGCCA)

Pseudo-Code for find-middle

```
function find-middle(S[1..m], T[1..n])
    DP-opt-cost(S[1..m/2], T[1..n]) -> F;
    DP-opt-cost(S[m..m/2], T[n..1]) -> reverse(F) -> B;
    for j = 1 to n: Z[j] = F[j] + B[j];
    return j such that Z[j] is the smallest;
end function
```

- Time complexity: O(mn)
- Space complexity: O(m+n)

Analysis of Hirschberg's Algorithm

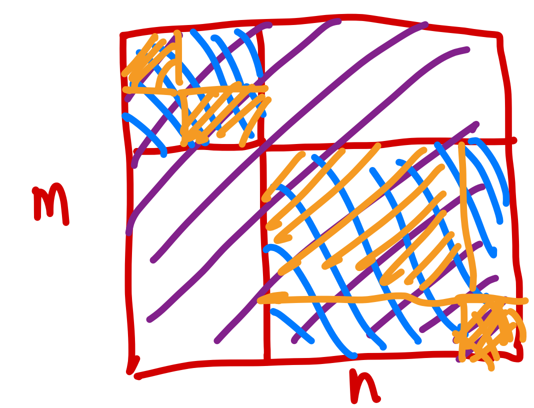
- Let T(m,n) be the running time, m = |S|, n = |T|
- Recurrence: T(m, n) = O(mn) + T(m/2, j*) + T(m/2, n-j*)

cmn

$T(m,n) \le 2cmn$

$$T(m,n) = cmn + 2c \cdot \frac{m}{2} \cdot j^* + 2c \cdot \frac{m}{2} \cdot (n - j^*)$$

$$= cmn + 2c \cdot \frac{m}{2} \cdot n = cmn + cmn \le 2cmn$$



Analysis of Hirschberg's Algorithm

- Let $P(m,n)$ be the space complexity, $m = |S|$, $n = |T|$
- Recurrence: $P(m, n) = \max\{O(m+n), P(m/2, j^*), P(m/2, n-j^*)\}$

$$\Rightarrow P(m,n) = O(m+n).$$