COMP3821/COMP9801 Workshop Week 10

Dynamic Programming

Jeremy Le and Xueyi Chee

2025T3

Announcements

- Fill out your MyExperience survey for your experience of us!
- Problem Set 9 Submission due on Friday 11:59pm.
  - Submissions on Gradescope.
- Project Report due Sunday 11:59pm.
- Poster session Friday 2pm - 5pm.
- ~~Help sessions on Friday 11am - 1pm in K17 G05.~~

Guided Problem 1

Guided Problem

Let $S[1..n]$ be a string of n characters. A *palindromic subsequence* is a subsequence of S that forms a palindrome. Recall that a palindrome that is read the same forwards and backwards, such as KAYAK. Describe and analyse a polynomial-time algorithm that returns the length of the longest palindromic subsequence of S.

For example, the length of the longest palindromic subsequence of DYNAMICPROGRAMMING is 7, as underlined.

→ 10 | 0 0 | 0 ···

$O(n^2)$ $O(n)$

$O(2^n)$ $O(n)$

Guided Problem 1 - Working

$f(\bullet) =$ longest palindromic subsequence of $\bullet$

subsequence

2) $f(\bullet) =$ longest palindromic subseq... of $\bullet$

substring

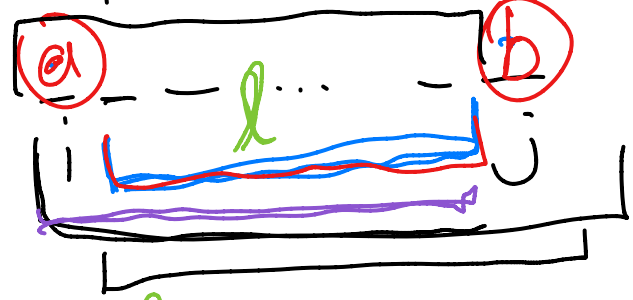
substring

$O(n^2)$

Guided Problem 1 - Working

$f(i,j)$ = longest pali... subse... of $S[i..j]$

S



a b a b

$f(i,j) = l + 2$ $\quad S[i] = S[j]$

$j-1-(i+1) = j-1-2$

or not

$j-i$

$$f(i,j) = \begin{cases} 2 + f(i+1, j-1) & S[i] = S[j] \\ \max(f(i, j-1), f(i+1, j)) & \text{or not} \end{cases}$$

$j-(i+1) = j-i-1$

Guided Problem 1 - Working

$f(i, j)$

$f(1, n) =$

a b $\quad f(i, i+1) = 3$

$f(i, i) = 1$

a a

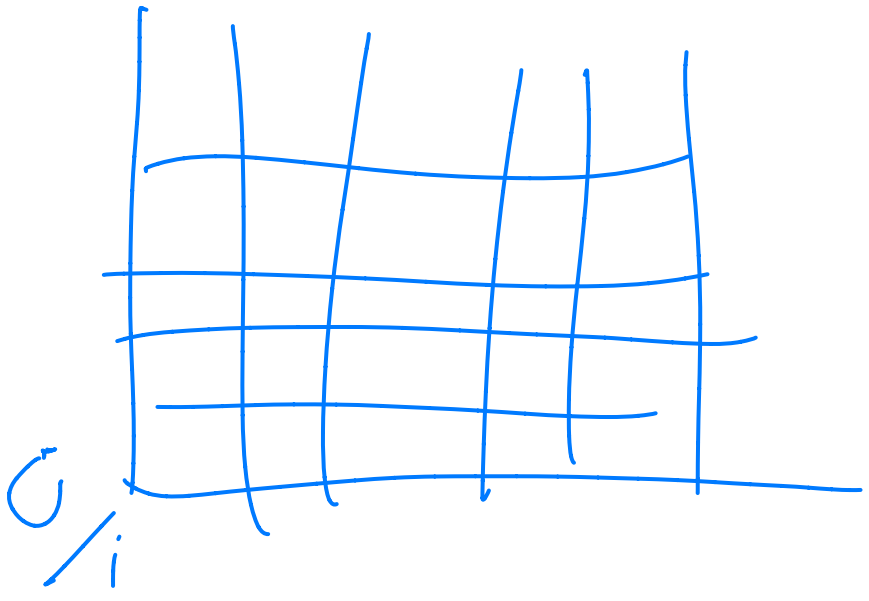
$f(i+1, i) = 0 \quad f(1, 2) =$

$f(2, 1) + 2 =$

Guided Problem 1 - Working

Order of computation:

increasing order of $j - i$



$f(i, j)$ $i < j$

$1 \leftrightarrow n$ $1 \leftrightarrow n$

$O(n^2)$ $O(1)$

$= O(n^2)$

Guided Problem 2

Guided Problem

The city of UNSW has decided to partition High Street into voting districts. There are n houses on High Street, living on consecutive addresses labelled $1, 2, \ldots, n$. Each voting district is an interval of addresses $i, i+1, \ldots, j$ with $1 \leq i \leq j \leq n$. The city has enforced two constraints.

- Each house on High Street must belong to *exactly* one voting district.
- The number of houses in each district must lie between k and $2k$, where k is a parameter.

Guided Problem 2 Cont.

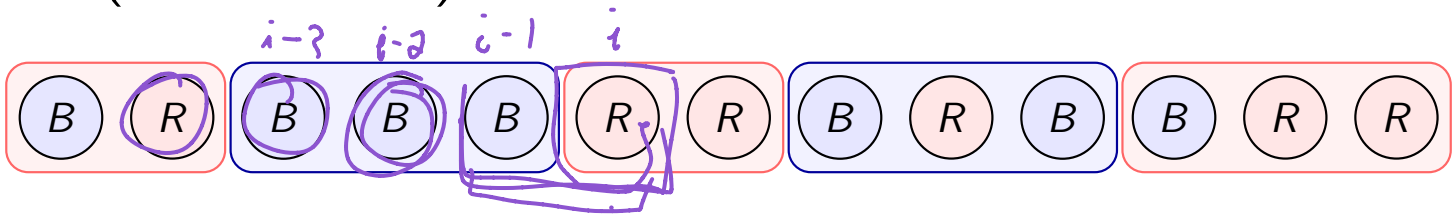
Guided Problem

In each election, houses can vote for either OCEANIA or EURASIA but they can only vote for one party. Since the majority of houses in the city of UNSW support OCEANIA, the city deems a district *favourable* if more than half of the houses in the district support OCEANIA; otherwise, the district is *unfavourable*. Of course, the city has a complete record of all of the votes in the election.

The next election is coming up and the city has decided to ask you, as the budding algorithmist, to help assign districts. You are given a boolean array VOTE$[1..n]$, where VOTE$[i] = 1$ if resident i voted for OCEANIA and VOTE$[i] = 0$ if resident i voted for EURASIA. You are also given an integer k. Describe and analyse a polynomial-time (in terms of k and n) algorithm that returns the largest number of favourable districts in any legal partition of the houses.

Guided Problem 2 Example

For example, consider the following legal partition of $n = 13$ houses with $k = 2$ (therefore, each district can be assigned 2 to 4 houses). Houses who voted for OCEANIA are coloured blue and the houses who voted for EURASIA are coloured red. There are two favourable districts (coloured blue) and three unfavourable districts (coloured red).



A legal partition of houses to five districts. The first district is unfavourable, the second district is favourable, the third district is unfavourable, the fourth district is favourable, and the fifth district is unfavourable.

Guided Problem 2 - Working

$f(i,j)$ represent largest number of districts between house i and j

For all $1 \le i \le n$

Districts(i) denotes the maximum number of favorable districts of any legal partition of $\{1, \ldots, i\}$

Recurrence for $i > k$

$$\text{Districts}(i) = \max_{1 \le j < i,\ k \le i-j+1 \le 2k} \{ \text{Districts}(j-1) + \text{GoodDistrict}(j, i) \}$$

$\uparrow O(k)$ $\quad \uparrow O(1)$ $\quad \uparrow O(k)$ $\Rightarrow O(k^2)$

GoodDistricts(i,j) houses from i to j is favorable

isFavorable by just summing or counting

Guided Problem 2 - Working

Base Case $1 \ldots k-1$

$Districts(i) = -\infty$ for $0 < i < k$

$Districts(i) = 0$ for $i = 0$

Final Solution $Districts(n)$

Order of computation Increasing order of i

Time Complexity # subproblems x how long each takes

$O(n)$ $O(k^2)$

$O(nk)$ $\Rightarrow O(nk^2)$

Guided Problem 2 - Working

$A = [12, 10, 3, 1, 2]$

$A = [10, 10, 3, 1, 2]$

$1 \quad 0$

$pre[i] = pre[i-1] + A[i]$

$pre = [\ 12, 22, 25, 26, 28\ ]$ ← $O(n)$

$O(\log n)$ updates and queries

$\sum_{i=2}^{u} A[i] = pre[u] - pre[\ ]$ ← $O(1)$

Range Tree

Segment Tree

$$= \sum_{i=1}^{u} A[i] - \sum_{i=1}^{\ } A[i]$$

$$= \sum_{i=2}^{u} A[i]$$

Guided Problem 2 - Solution

For each $1 \leq i \leq n$, let $\textsc{Districts}(i)$ denote the maximum number of favourable districts in any legal partition of the set $\{1, \ldots, i\}$.

- **Recurrence.** For any pair of indices $i < j$, we can compute $\textsc{GoodDistrict}(i, j)$ that returns 1 if the majority of residents from house i to j voted for $\textsc{Oceania}$ and 0 otherwise.
  The subproblem satisfies the following recurrence,

$$\textsc{Districts}(i) = \max_{\substack{1 \leq j < i, \\ k \leq i-j+1 \leq 2k}} \{\textsc{Districts}(j-1) + \textsc{GoodDistrict}(j, i)\}.$$

Guided Problem 2 - Solution

- **Base case.** To ensure that we obtain legal partitions, we set

$$\textsc{Districts}(i) = \begin{cases} 0 & \text{if } i = 0 \\ -\infty & \text{if } 0 < i < k. \end{cases}$$

- **Final solution.** Occurs at $\textsc{Districts}(n)$.
- **Order of evaluation.** Increasing order of i.
- **Time complexity.**
  - $O(n)$ many subproblems.
  - For each fixed i, we iterate over $O(k)$ many indices for j.
  - for each fixed pair i, j such that $j < i$, we compute $\textsc{GoodDistrict}(i, j)$ in $O(k)$ time.
  - Each subproblem takes $O(k^2)$ time.
  - Overall time complexity $O(nk^2)$.
- *Note.* We can compute $\textsc{GoodDistricts}$ quicker by first computing prefix sums. Reducing time complexity to $O(kn)$.