EECS3101 notes::Dynamic programming examples

Jerry Wu

2023-02-28

Week 7 practice

Example 1:: Optimizing an itinerary

The details of the problem are as follows::

- We wish to travel from city 0 to city n using buses.
- The cost of going from i to j is defined as $C_{i,j}, \forall i, j (i < j)$.
- All buses only go from a lower numbered city to a higher numbered city.
- What is the minimum cost of going from 0 to n.

Solution

> *"If you have an insecurity about your final answer, you might have some reviewing to do."-Larry YL Zhang 2023*

1. List out the subproblems
   What is the minimum cost of going from i to j? What about i to n or 0 to i? We can try the last one because the main problem is from 0 to n. Let $ans[i] = minCost(0 \to i)$

2. Find a recursive relation between the subproblem solutions. Consider an intermediate value between 0 and j defined as $k \in [0, j-1] \equiv k \in [0, j)$ which is a node directly **preceding** j.

   $$ans[j] = \min_{\forall k \in [0,j)} \{ans[k] + C_{k,j}\}$$

   With this, we're able to find the minimum of all the minimums (the absolute minimum). This search is exhaustive.

   > *"Don't overthink it. It should be straightforward."-Larry YL Zhang 2023*

3. Data structure to store the answers?
We have a problem that is $O(n) = \Theta(n)$ because $j \in [0, n]$, so our runtime for the calculation is also $O(n) = \Theta(n)$. So the overall runtime would be $\Theta(n^2)$. Because we have n cities, we can safely say that the space complexity is $\Theta(n)$.

Base case

For the base case, we can just set $k = 0$. It is the case where we don't move to any new city.

$$ans[0] = ans[0] + C_{0,0} = 0$$

The code

So now, we can code our solution.

```
CheapestItinerary(C)
{
    ans = array of size n+1;
    ans[0] = 0;
    for(i = 1; i <= n; i++)
    {
        ans[i] = +infinity; //unvisited node, set to a large value
        for(k = 0; k < i; k++)
        {
            ans[i] = min(ans[i], ans[k] + C[k,i])
        }
    }

    return ans[n];
}
```

Example 2:: Matrix chain multiplication

Suppose we have two arbitrary matrices::

- A is an $n \times m$ matrix
- B is an $m \times k$ matrix
- $C = AB$ which is an $n \times k$ matrix. produced via $n \times m \times k$ scalar multiplications.

  "Quick maths"-Larry YL Zhang 2023

How many scalar multiplications are needed with different parenthisization? Compute $A_1 A_2 A_3 \dots A_n$ with the **fewest** number of multiplications.

The size of A_1 is $d_0 \times d_1$, A_2 is $d_1 \times d_2$, A_n is size $d_{n-1} \times d_n$. So the input is an array of d_0 to d_n where size is $n + 1$.

Solution

We need to store the subproblems in a 2d array. How should we populate this 2d array to get the optimal solution? Well, we need to make sure that RHS is available before we can evaluate LHS, i.e. $ans[i][j]$ with a smaller $|j - 1|$ must be evaluated first. So we can populate the 2d array diagonally, where the base cases are all the indices directly down the diagonal: $((0,0),(1,1),(2,2),(3,3)...(n,n))$. Illustration is available on **slide 19**.

The code

```
//input d is an array of size n+1
//we are storing the dimensions of the matrices in the array

int matrixMultiplication(int[] d)
{
    int[][] ans = a 2d array of size (n+1)x(n+1)
    for(i = 1; i <= n; i++)
    {
        ans[i][i] = 0 //populate each element in the diagonal to be 0 (the base cases)
    }

    //diff is the difference between i and j

    for(diff = 1; diff <= n; diff++)
    {
        for(i = 1; i <= n; i++)
        {
            j = i + diff;
            ans[i][j] = +infinity
            for(k = i; k < j; k++)
            {
                ans[i][j] = min(ans[i][j], ans[i][k] + ans[k+1][j] + d[i-1]*d[k]*d[j])
            }
        }
    }
    //the optimal solution will always be in the first row.
    return ans[1][n]
}
```