3121期末复习讲义1

复习重点

1. divide and conquer
2. FFT
3. Greedy
4. dynamic programming
5. max flow

4题 3小时 （12小时里任选）

期末课安排：

周一part1 to part3 （2.5h）

周二part4 to part5 （3h）

周三 sample+答疑 （2.5h）

Divide and Conquer

Big O notation (Theta, Omega)

Master theorem

Let:

- a ≥ 1 be an integer and and b > 1 a real;
- f(n) > 0 be a non-decreasing function;
- T (n) be the solution of the recurrence **T (n) = a T (n/b) + f (n);**

Then:

1. If $f(n) = O(n^{\log_b a-\varepsilon})$ for some ε > 0, then $T(n) = \Theta(n^{log_b a})$;
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \log_2 n)$;
3. If $f(n) = \Omega(n^{\log_b a+\varepsilon})$ for some $\varepsilon > 0$, and for some **c<1** and some n_0, $af(n/b) \leq cf(n)$ holds for all $n > n_0$, then $T(n) = \Theta(f(n))$;
4. If none of these conditions hold, the Master Theorem is NOT applicable.

（不用背，考试考到记得对着抄就行）

Divide and Conquer

分治算法，核心思想：

将一个难以直接解决的大问题，分割成一些规模较小的相同问题，以便各个击破，分而治之。

分治法在每一层递归上都有三个步骤：

step1 分解：将原问题分解为若干个规模较小，相互独立，与原问题形式相同的子问题；

step2 解决：若子问题规模较小而容易被解决则直接解，否则递归地解各个子问题

step3 合并：将各个子问题的解合并为原问题的解。

实际上就是类似于数学归纳法，找到解决本问题的求解方程公式，然后根据方程公式设计递归程序。

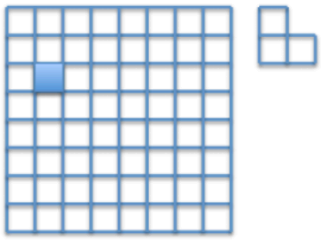
1、一定是先找到最小问题规模时的求解方法

2、然后考虑随着问题规模增大时的求解方法

3、找到求解的递归函数式后（各种规模或因子），设计递归程序即可。

例1 （不是那么容易看出来的但实际上就是分治思想的题）

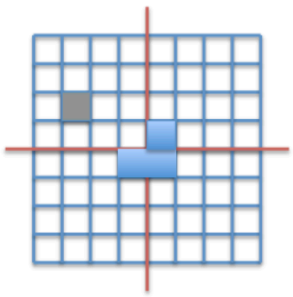
You are given a 2^n × 2^n board with one of its cells missing (i.e., the board has a hole); the position of the missing cell can be arbitrary. You are also given a supply of “dominoes” each containing 3 such squares; see the figure below. Your task is to design an algorithm for covering the entire board with such



“dominoes”, except for the hole which should remain uncovered.

Solution: We can do this recursively / inductively. This problem is easy for $n = 1$: we are given a 2×2 board with 1 square removed, so we can just fit our "domino" into the remaining 3 squares.

Now suppose $k \geq 1$ is an integer, and we have a solution for *all* such $2^k \times 2^k$ boards that are missing exactly one square. If we are given a $2^{k+1} \times 2^{k+1}$ board, we can split it into its four quadrants, giving four $2^k \times 2^k$ boards:



Our missing square necessarily occurs in one of these quadrants, so we tile that recursively using our algorithm. For the remaining three quadrants, we

place a "domino" on the three squares that are touching the centre (the place where we split our boards), and treat those squares as removed. We can then apply our algorithm recursively into these three quadrants as well, as they too are now missing a square.

例2：（函数题）

Assume you are given two arrays A and B, each containing n distinct integers and equation $x^8 - x^4y^4 = y^6 + x^2y^2 + 10$. Design an algorithm which runs in time O(n log n) which finds if A contains a value for x and B contains a value for y that satisfy the equation.

把两个未知数的等式转化成一个未知数的方程

用二分法找**单调函数**的解

Solution: First, observe that all the terms are even powers, so we can ignore the sign of the numbers in our arrays. Then, we can rewrite the equation as $y^6 + x^4y^4 + x^2y^2 = x^8 - 10$. Now, for a fixed value of x, the left-hand side is non-decreasing with respect to y (for non-negative y) and the right-hand side is constant. This gives rise to a familiar solution: we sort B, say, into non-decreasing order. We iterate through A, giving us our possible values for x. For a fixed value of x, we binary search over B to check if a feasible y exists: to determine if we want smaller or larger values for y, we compare $y^6 + x^4y^4 + x^2y^2$ with $x^8 - 10$,and choose the correct half of our current binary search's range to continue with.

例3: 找最大/最小(回忆之前讲过的找最大和第二大的题)

Assume you have an array of 2n distinct integers. Find the largest and the smallest number using 3n − 2 comparisons only.

Solution: We split all numbers in 2n/2 = n pairs and compare the two numbers in each pair. we put all the smaller ones on pile S and all the larger ones in pile L. Clearly the smallest among all numbers cannot be in pile L and the largest among all numbers cannot be in pile S. Thus we now find the smallest number in S using n − 1 additional comparisons and the largest number in M also with n − 1 comparisons, in total using only 3n − 2 comparisons.

FFT

Complex roots of unity

Roots of unity of order n are complex numbers which satisfy $z^n = 1$.

If $z^n = |z|^n(\cos(n\arg(z)) + i\sin(n\arg(z))) = 1$ then $|z| = 1$ and $n\arg(z)$ is a multiple of 2π; Thus, $n\arg(z) = 2\pi k$, i.e., $\arg(z) = \frac{2\pi k}{n}$ We denote $\omega_n = e^{i2\pi/n}$; such ω_n is called a primitive root of unity of order n.

$$((\omega_n)^k)^n = (\omega_n)^{nk} = ((\omega_n)^n)^k = 1^k = 1.$$

for all k such that 0 ≤ k ≤ n − 1

- $\omega_n^k\omega_n^m = \omega_n^{k+m}$
- If $k + m \geq n$ then $k + m = n + l$ for $l = (k + m) \mod n$ and we have $\omega_n^k\omega_n^m = \omega_n^{k+m} = \omega_n^{n+l} = \omega_n^n\omega_n^l = 1 \cdot \omega_n^l = \omega_n^l$ where $0 \leq l < n$.
- The Cancelation Lemma:$\omega_{kn}^{km} = \omega_n^m$ for all integers k, m, n.

不用背，写到对着ppt套公式。

例1（最多乘法次数）

(DFT思想)

$P(x) = a_0 + a_2x^2 + a_4x^4 + a_6x^6$ and $Q(x) = b_0 + b_2x^2 + b_4x^4 + b_6x^6 + b_8x^8$ using at most 8 **multiplications** of large numbers;

1. 如果可以改写原函数让他们次项变小，改写。（P -》 degree 3 Q-〉degree 4）
2. 结果式的次项 （7）
3. 代点进去
4. 计算P(x) Q(x)

solution: First, observe that it is enough to be able to multiply any degree 3 poly-nomial by a degree 4 polynomial using 8 such multiplications: both P and Q are really polynomials with those respective degrees in x^2.

Since the result is a polynomial of degree 7, we can uniquely determine it by determining its values at 8 points. For this we can choose {−4, −3, . . . , 3} or alternatively, the 8th roots of unity. We evaluate P at these 8 points and Q at these 8 points: these are only multiplications of a large number my a (constant size) scalar, so these operations are cheap. We then multiply the results **pointwise**: these require precisely 8 large number multiplications. We can then determine the coefficients from these values by setting up a system of linear equations. Solving this is done by inverting a constant matrix (as described in the course), so this inversion can even be done by hand, offline and requires no computation. We then multiply the matrix by the vector formed by the pointwise multiplications, which again only multiplies these results by scalars, to give the final polynomial.

变形

（数学技巧）

Multiply two complex numbers z1 = a + i b and z2 = c + i d using only 3 real number multiplications.

solution:

We know that z1z2 = (ac − bd) + i(bc + ad). Now we can perform the multiplication (a + b)(c + d) = ac + bd + (bc + ad). With a further two multiplications, we can find ac and bd and hence the real part of z1z2. By subtracting both of these from (a + b)(c + d), we obtain the imaginary part, too, so we can find the product using three multiplications.

例2 （关于unity root的计算）

Describe all k which satisfy $i\omega_{64}^{13}\omega_{32}^{11} = \omega_{64}^{k}$ (i is the imaginary unit).

$$\omega_{64}^{k} = \omega_{64}^{16}\omega_{64}^{13}\omega_{64}^{22} = \omega_{64}^{51}$$

k = 51 + 64 n

例3 （Convolution）

Describe how you would compute all elements of the sequence F(0), F(1), F(2), . . . , F(2n) where

$$F(m) = \sum_{\substack{i+j=m \\ 0\le i,j\le n}} \log(j+2)^{i+1}$$

in time O(nlogn).

Solution: We have that

$$F(m) = \sum_{\substack{i+j=m \\ 0 \le i,j \le n}} \log\left((j+2)^{i+1}\right)$$
$$= \sum_{\substack{i+j=m \\ 0 \le i,j \le n}} (i+1)\log(j+2)$$

Now this is just a convolution of two sequences. We can set up the following polynomials

$$P(x) = \sum_{i=0}^{n} (i+1)x^i$$
$$Q(x) = \sum_{j=0}^{n} (\log(j+2))x^j$$

and calculate their product, $(PQ)(x)$ in $O(n \log n)$ time using the Fast Fourier Transform (FFT). Then $F(m)$ is just the coefficient of x^m in $PQ(x)$.

Greedy

贪心算法，又名贪婪法，是寻找**最优解问题**的常用方法，这种方法模式一般将求解过程分成**若干个步骤**，但每个步骤都应用贪心原则，选取当前状态下**最好/最优的选择**（局部最有利的选择），并以此希望最后堆叠出的结果也是最好/最优的解。

贪婪法的基本步骤：

步骤1：从某个初始解出发； 步骤2：采用迭代的过程，当可以向目标前进一步时，就根据局部最优策略，得到一部分解，缩小问题规模； 步骤3：将所有解综合起来。

证明optimal的方法：

1. 假设有一个最优算法和我们的不一样，证明出该算法可以在有限步骤内转换为我们的算法。（调换的步骤类似Bubble sort）
2. 反证法。假设有算法比我们更优，得出矛盾。

例1 （硬币问题）

In Elbonia coin denominations are 81c, 27c, 9c, 3c and 1c. Design an algorithm that, given the amount that is a multiple of 1c, pays it with a minimal number of coins**. Argue that your algorithm is optimal.**

solution：

We use the greedy strategy, always giving the largest number of the highest possible denomination. Thus we first give bx/81c many 81c coins and continue with the 27c coins etc.

To prove that this results in the fewest possible number of coins assume the opposite, that there is a combination with fewer coins for the same amount and among such combinations pick the one using the fewest number of coins. Such combination of coins clearly is not the one produced by our strategy. Without loss of generality, we can assume that such a combination violates the greedy strategy already with the number of 81c coins used. Thus, after subtracting the amount given with 81c coins, we are left with an amount of at least 81c. But such a combination can contain at most two 27c coins, otherwise it would not be optimal: if three or more 27c coins were present we could replace three of them with a single 81c coin, thus violating our assumption that such a combination employs the minimal possible number of coins. However, after subtracting at most 2 × 27c we end up with an amount of at least 27c to be given using only 9,3 and 1c coins. Again, at most 2 coins of 9c could be used and we end up with at least 9c to be given using only 3 and 1 cent coins. At most two 3 cent coins can be used and we end up with 3c to be given using only 1c coins, which clearly violates the assumed optimality of such a combination.

拓展：什么样的硬币组合可以用greedy，什么样的不能?

81c, 27c, 9c, 3c and 1c

n > 81

81 + (27, 9,3, 1)

one

1 3 7

时间表问题（asst题）

- sort by ending time

例2: stalls and board

There is a line of 111 stalls, some of which need to be covered with boards. You can use up to 11 boards, each of which may cover any number of consecutive stalls. Cover all the necessary stalls, while covering as few total stalls as possible.

Solution: Let us call the stalls which need to be covered the *necessary stalls*. Cover all stalls between (and including) the first and the last necessary stall with a single board. Look for the longest contiguous sequence of stalls which are not necessary and excise the corresponding slice of the board. Continue in this manner until you obtain 11 pieces. Optimality is easy to prove as before by assuming there is a better way of covering the necessary stalls, take the best possible such and look for the longest stretch of stalls which are not necessary but remain covered in this “better solution”. Remove this slice but join two pieces with a smaller gap to obtain an even better solution and achieve a contradiction.

例3 工厂类optimal的第一种证明

You are running a small manufacturing shop with plenty of workers but with a single milling machine. You have to produce n items; item i requires mi machining time first and then pi polishing time by hand. The machine can mill only one object at a time, but your workers can polish in parallel as many objects as you wish. You have to determine the order in which the objects should be machined so that the whole production is finished as quickly as possible. Prove that your solution is optimal.

solution：

Observe that our solution is uniquely determined by the order which we choose to mill the items, and that no matter which order we choose, the total milling time is constant. Specifically, the production is finished when the last item has finished polishing. We sort our items in non-increasing order of pi and mill them in this order. We now prove that this is optimal.

Suppose, without loss of generality, that our items are in order 1, 2, . . . , n. Then define the finishing time of item k as

$$f(k) = p_i + \sum_{j=1}^{i} mi$$

so our overall finishing time is the maximum among all f(k). Further suppose there are two adjacent items i and i + 1 in our order such that pi < pi+1: if there aren't, then the items are already in sorted order. Then if we process our items in the modified order 1, 2, . . . , i − 1, i + 1, i, i + 2, i + 3, . . . , n and let g(k) be the finishing time for item k under this order. Then, g(k) = f(k) whenever k not∈ {i, i + 1}. Now let S = m1 + m2 + · · · + mi−1. Then we have that

f(i + 1) = S + mi + mi+1 + pi+1 g(i) = S + mi+1 + mi + pi g(i + 1) = S + mi+1 + p_{i+1}

So since $p_{i+1} > pi$, we have f(i + 1) > g(i) and since all milling times are positive, we also have that f(i + 1) > g(i + 1). Hence, swapping i and i + 1 gives a modified order that is never worse than our original order, which never increases maxi{f(i)}. We can repeat this argument until we get sorted order: to show that this indeed occurs, observe that this procedure is the same as bubble sort, for which we know from earlier courses that always terminates.

有sort什么东西的时候，我们就可以尝试用第一种方法证明optimal

例4 从直线到圈

asst火车题

You are given a set S of n overlapping arcs of the unit circle. The arcs can be of different lengths. Find a largest subset P of these arcs such that no two arcs in P overlap (largest in terms of total number of elements, not in terms of total length of these arcs). Prove that your solution is optimal.

Solution: Recall that this problem is an easy greedy on a line (pick the one that ends first, repeat). For each arc, we separately assume it is in the solution, then repeat clockwise along the arc just as in the line. Then, take the maximum among all these solutions. Note that this solution runs in O(n2)time. To prove that this solution is correct, we note that we clearly must choose some arc to be a part of our set. Once this arc is chosen, we cannot take any arc intersecting with this one, so we effectively "delete" the arc from the circle. This leaves a line, and we can appeal to the proof of correctness for our "line-algorithm" in this case.

关键：把圆变回直线

变：

You are given a set S of n overlapping arcs of the unit circle. The arcs can be of different lengths. You have to stab these arcs with minimal number of needles so that every arc is stabbed at least once. In other words, you have to find a set of as few points on the unit circle as possible so that every arc contains at least one point. Prove that your solution is optimal.

Solution: Recall that this problem is also an easy greedy on a line (pick the first ending point, place a needle there, remove all stabbed intervals, repeat). Again, we can reduce this problem to a line as follows: pick a starting interval and stab it at its end. Remove all stabbed intervals and pick among the rest an interval with the "earliest" end and stab it at the very end. Continue in this way until all intervals have been stabbed or eliminated and record the number of pins used. Repeat this procedure for all intervals as the starting interval. Among all thus obtained solutions pick the one which uses the fewest number of needles. **To prove that this solution is correct, take an optimal solution and move all needles to the right till they stab one of the stabbed intervals at its right end. Pick any such interval stabbed at its right end; then such solution would have been obtained by our method while stating with that interval.**

假设一个最优解（有最少的针的个数）

通过移动"针"的位置（右移）得到我们的解法

因此我们的解法不会比最优解用的针更多

因此我们的解法正确。

例5 图论+贪心

In Elbonia cities are connected with one way roads and it takes one whole day to travel between any two cities. Thus, if you need to reach a city and there is no a direct road, you have to spend a night in a hotel in all intermediate cities. You are given a map of Elbonia with tool charges for all roads and the prices of the cheapest hotels in each city. You have to travel from the capital city C to a resort city R. Design an algorithm which produces the cheapest route to get from C to R.

Solution: Assuming all tolls and hotel costs are non-negative, we can use Dijkstra's algorithm for this problem: there are many ways to see this, we provide just one. We can split any city u into two nodes, uin and uout. All roads u → v will be uout → vin in our new graph, and vice-versa. The weight of each such edge is equal to the toll cost of that road. Additionally, for all u we have an edge uin → uout with weight equal to the cost of the cheapest hotel in u. Then, the shortest route from C to R is given by the shortest route from Cout to Rin in our modified graph.

复习一下 Dijkstra's algorithm，了解这个算法是干嘛的，怎么做的。

例6 （climb up and down）

Assume that you got a fabulous job and you wish to repay your student loan as quickly as possible. Unfortunately, the bank "National Road Robbery" which gave you the loan has the condition that you must start by paying off $1 and then each subsequent month you must pay either double the amount you paid the previous month, the same amount as the previous month or a half of the amount you paid the previous month. On top of these conditions, your schedule must be such that the last payment is $1. Design an algorithm which, given the size of your loan, produces a payment schedule which minimizes the number of months it will take you to repay your loan while satisfying all of the banks requirements.

Solution: Suppose the loan is N dollars. We can view the powers of two that we choose in the following way. Let 2^k be the largest amount we pay in any single month. Then, we necessarily must have paid $2^0+2^1+\cdots+2^k$ "on the way up" and then an extra $2^{k-1}+2^{k-2}+\cdots+2^0$ "on the way down". This means we pay a minimum of $2(2^0+2^1+\cdots+2^{k-1})+2^k = 2(2^k-1)+2^k = 3\cdot 2^k-2$ so necessarily $N \geq 3\cdot 2^k-2$. We choose the largest k that satisfies such an inequality. Note that this inequality is equivalent to $(N+2)/3 > 2^k$, which means that $k = \lfloor \log_2((N+2)/3) \rfloor$. Note also that, by maximality of k we also have $N < 3\cdot 2^{k+1}-2$ which means that $N < 3\cdot 2^k - 2 + 3\cdot 2^k$, i.e., $N-(3\cdot 2^k-2) < 3\cdot 2^k$. Thus, the additional amount A you have to pay satisfies $A < 3\cdot 2^k$.

- If $A = 0$ you are done; if $1 \leq A < 2^k$ you represent A in binary and see what powers of 2 you have to pay twice, either on your way up to 2^k or on your way down.
- If $2^k \leq A < 2^{k+1} = 2\cdot 2^k$ you pay the amount 2^k twice (in total) and then you are left with an additional amount $0 \leq B \leq 2^k-1$ which you pay as in the previous case.
- If $2^{k+1} = 2\cdot 2^k \leq A < 3\cdot 2^k$ then you pay 2^k three times in total and you are left with an additional amount to pay $0 \leq C < 2^k$ which you again represent in binary and see which of the amounts from $1 = 2^0$ to 2^{k-1} you have to pay twice.

Proof of the correctness: well it should be obvious from the algorithm description, but you can try to give a more rigorous proof, similar to the proof of optimality of the greedy giving change in coins algorithm.