Alg_Answers

不包含编程题。

Assignment1

1.1

题目

1 Divide and Conquer

You are interested in analyzing some hard-to-obtain data from two separate databases. Each database contains n numerical values, so there are $2n$ values total and you may assume that no two values are the same. You'd like to determine the median of this set of $2n$ values, which we will define here to be the n^{th} smallest value.

However, the only way you can access these values is through *queries* to the databases. In a single query, you can specify a value k to one of the two databases, and the chosen database will return the k^{th} smallest value that it contains. Since queries are expensive, you would like to compute the median using as few queries as possible.

Give an algorithm that finds the median value using at most $O(\log n)$ queries.

解答

算法描述

原问题为在两个提供第k小查询的序列中找出一个中位数。可以将两个序列看成有序序列，从而将问题变成标记问题，即将前n小的数字在两个序列中标记出来，也就是将前n小的数字恰当的安排在两个序列中，更确切的目的就是找出构成前n小的两个子序列在两个序列中的终点，较大的那个便是两个序列的中位数。 设序列为 $a1$,$a2$，长度均为n，可以通过二分查找确定前n小的序列在$a1$中的终点来求解此问题。

伪代码

```
L = 1, R = n - 1;
while(L < R)
    mid = (L + R) / 2;
    //判断条件 即a2中的终点的下一个数字应该要大于a1中的终点
    if (getKth(a2, n - mid + 1) < getKth(a1, mid))
        R = mid - 1;
    else
        L = mid;
    end if
end while
// 2n个数字中位数应该有两个 这里不做特判， 输出较大的那个
answer = max(getKth(a1, L), getKth(a2, n - L))
```

算法正确性证明 因为问题存在偏序，所以使用二分查找法可以得到正确的解。 **算法复杂度** 使用了二分查找，故复杂度为$O(logn)$

1.2

题目

2 Divide and Conquer

Find the k^{th} largest element in an unsorted array. Note that it is the kth largest element in the sorted order, not the k^{th} distinct element.

INPUT: An unsorted array A and k.

OUTPUT: The k^{th} largest element in the unsorted array A.

解答

算法描述 求解第k大可以使用类似快速排序的分治算法。选取一个基准数字x，用类似快排的方式将数组分成两部分，记为a,b，然后分情况讨论，如果$size(a) >= k$，则对a进行递归操作，否则递归求解b的第$k - size(a)$大即可。

伪代码

```
int getKth(array a, int k, int L, int R)
x = a[L + R >> 1]
l = L, r = R
while (l < r)
    while (a[l] <= x && l < r) l++;
    while (a[r] > x && l < r) r--;
    if (l < r) swap(a[l], a[r]);
end while
if (l == k) return x;
if (l > k)
    return getKth(a,k,L,l);
else
    return getKth(a,k-l,l + 1,R);
```

算法正确性证明 快速排序很像从上而下的二叉排序树，我们在找第k大的过程就是在找排序树中的第k个节点，由于子树中的节点是有序的，对于不包含目标节点的子树仅仅需要他的大小来计数，这和快速排序中的partition操作十分类似。由于使用二叉排序树可以正确的找到第k大，所以用类似快排的方式也可以。

复杂度分析 虽然快排的平均复杂度为$O(nlogn)$，但是求解第k大问题时问题的规模在不断地缩小，复杂度小于$O(nlogn)$,但是另一方面，最坏情况下需要$\sum_{i=0}^{log(n)} log(n \cdot (\frac{1}{2})^i) \cdot n \cdot (\frac{1}{2})^i$次计算，本算法多数情况下的效率是渐进$O(N)$的。

1.3

题目

3 Divide and Conquer

Consider an n-node complete binary tree T, where $n = 2^d - 1$ for some d. Each node v of T is labeled with a real number x_v. You may assume that the real numbers labeling the nodes are all distinct. A node v of T is a *local minimum* if the label x_v is less than the label x_w for all nodes w that are joined to v by an edge.

You are given such a complete binary tree T, but the labeling is only specified in the following *implicit* way: for each node v, you can determine the value x_v by *probing* the node v. Show how to find a local minimum of T using only $O(\log n)$ *probes* to the nodes of T.

解答

只是找到一个即可，比如在根节点上，如果两个子节点都比自己大，那么返回根节点，否则，递归进入比根节点小的那个子树上（如果两个都比根节点小取任意一个，都可以得到答案）这样我们最终会停止在两种情况下，一种是得到了一个两个子节点都比自己大的节点，这样返回答案，另一种是到了树的叶子节点，可以证明，如果两个叶子节点都比他们的父节点大，那么他们的父节点是我们的答案，否则至少有一个叶子节点是我们的答案。

伪代码

```
get_local_min(Tree T)
  if (T.Lchild == null) return T;
  if (T.Lchind.val < T.val) get_local_min(T.Lchild);
  if (T.Rchind.val < T.val) get_local_min(T.Rchild);
  return T;
```

正确性证明

如解答中所说，算法一定会停止在一个合法的非叶子节点上或者父节点比自己大的叶子节点上。而这两种情况一定有一种成立，所以算法是正确的。

复杂度分析

算法从根节点不回溯的执行到叶子节点，所以复杂度为$O(logN)$

1.4

题目

4 Divide and Conquer

Suppose now that you're given an $n \times n$ grid graph G. (An $n \times n$ grid graph is just the adjacency graph of an $n \times n$ chessboard. To be completely precise, it is a graph whose node set is the set of all ordered pairs of natural numbers (i, j), where $1 \leqslant i \leqslant n$ and $1 \leqslant j \leqslant n$; the nodes (i, j) and (k, l) are joined by an edge if and only if $|i - k| + |j - l| = 1$.)

We use some of the terminology of problem 3. Again, each node v is labeled by a real number x_v; you may assume that all these labels are distinct. Show how to find a local minimum of G using only $O(n)$ probes to the nodes of G. (Note that G has n^2 nodes.)

解答

每次找到中间行和中间列的最小值，然后判断是否合法，如果不合法则找到这个最小值的最小合法邻居(如果在边界上要找两次邻居，并始终检查合法性)，然后将问题的规模缩小到这个邻居所在的1/4象限中。

伪代码

```
//从(x1,y1)到(x2,y2)的矩阵
Point get_local_min(Matrix mat, int x1, int y1, int x2, int y2)
{
    midx = x1 + x2 >> 1;
    midy = y1 + y2 >> 1;
    minval = min(val in mid cloumn and mid row);
    pos = minval.pos;
    //如果这个值是合法的则返回
    if (check(pos) == true)return pos;
    //否则找他的最小邻居并判断
    min_neighbor = minneighbor(pos);
    if (check(min_neighbor) == true)return min_neighbor;
    //如果这个点在中间行列上，要再求一次邻居弄到某个象限里面
    if(min_neighbor in mid row or mid cloumn)
    min_neighbor = minneighbor(min_neighbor);
    if (check(min_neighbor) == true)return min_neighbor;
    //如果不是解，就判断其对应象限并缩小范围进行查找
    if (min_neighbor.x < midx && min_neighbor.y < midy)
    return get_local_min(mat,x1,y1,midx,miny);
    if (min_neighbor.x < midx && min_neighbor.y > midy)
    return get_local_min(mat,x1,midy,midx,y2);
    if (min_neighbor.x > midx && min_neighbor.y < midy)
    return get_local_min(mat,midx,y1,x2,miny);
    if (min_neighbor.x > midx && min_neighbor.y > midy)
    return get_local_min(mat,midx,midy,x2,y2);
}
```

正确性证明

我们始终在访问一个递减序列，最终一定会再某个答案处终止。

复杂度分析

每次求最小值为O(N)，但是问题的规模在不断减小

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

所以复杂度为$O(N)$

1.5

题目

5 Divide and Conquer

Given a convex polygon with n vertices, we can divide it into several separated pieces, such that every piece is a triangle. When $n = 4$, there are two different ways to divide the polygon; When $n = 5$, there are five different ways.

Give an algorithm that decides how many ways we can divide a convex polygon with n vertices into triangles.

解答

递推问题，将所有的点编号1到n，公式为

$sum(n) = \sum_{i=2}^{n-1} sum(i) * sum(n+1-i)\ n >= 4$

$sum(3) = 1, sum(2) = 1$

伪代码

> 实现上面的公式即可，注意记忆化搜索加速

正确性证明

从多边形中抽出一个三角形，剩下两个更小的多边形，从而产生递推关系。凸N边形分为三角形总数sum(n)等于i边形的分法总数乘以n+1-i边形的分法总数之积，最后累加起来。

复杂度分析

使用记忆化搜索，每层的状态为N，总复杂度为$O(N^2)$

1.6

题目

6 Divide and Conquer

Recall the problem of finding the number of inversions. As in the course, we are given a sequence of n numbers $a_1, \cdots, a_n$, which we assume are all distinct, and we difine an inversion to be a pair $i < j$ such that $a_i > a_j$.

We motivated the problem of counting inversions as a good measure of how different two orderings are. However, one might feel that this measure is too sensitive. Let's call a pair a *significant inversion* if $i < j$ and $a_i > 3a_j$. Given an $O(n \log n)$ algorithm to count the number of significant inversions between two orderings.

解答

在原来的归并排序计算逆序数的基础上加一个特殊判断即可，例如当前合并序列为a,b，算法执行到了$a[i] > b[j]$,此时计算逆序数需要计算$b[j]$ 的三倍在a 中的位置，需要加上一个二分查找。

伪代码

核心部分

```
    if (a[x] < a[y])
        {
            tmp[start++] = a[x++];
        }
     else
        {
            int pos = upper_bound(a + x, a + mid + 1, 2 * a[y]) - a;
            cnt += mid - pos + 1;
            tmp[start++] = a[y++];
        }
```

其余部分和归并排序相同

正确性证明

略。。

复杂度分析

虽然加了一个查询，复杂度仍然为$O(NlogN)$

1.7

题目

7 Divide and Conquer

A group of n ghostbusters is battling n ghosts. Each ghostbuster is armed with a proton pack, which shoots a stream at a ghost, eradicating it. A stream goes in a straight line and terminates when it hits the ghost. The ghostbusters decide upon the following strategy. They will pair off with the ghosts, forming n ghostbuster-ghost pairs, and then simultaneously each ghostbuster will shoot a stream at his chosen ghost. As we all know, it is very dangerous to let streams cross, and so the ghostbusters must choose pairings for which no streams will cross. Assume that the position of each ghostbuster and each ghost is a fixed point in the plane and that no three positions are collinear.

1. Show that there exists a line passing through one ghostbuster and one ghost such the number of ghostbusters on one side of the line equals the number of ghosts on the same side. Describe how to find such a line in $O(n \log n)$ time.
2. Give an $O(n^2 \log n)$-time algorithm to pair ghostbusters with ghosts in such a way that no streams cross.

解答

7.1

任选一个敢死队员，以此敢死队员为基准，将所有的点按照极角排序，因为题目保证没有三点共线所以这个顺序是唯一的，极角排序复杂度$O(nlog(n))$，之后在这个序列上进行线性扫描，找到一个位置，使得这个位置（包括这个位置）之前的两种点个数相同，此时找到了一条可以将点等分的线。

7.2

在上述算法基础之上，每次找到一条线，就记录下当前的这条边，并且将这两个点删除，再重新选取点，直到所有点都找到了对应的点。

伪代码

7.1

```
getLine(PointArray busters, ghosts)
    PointArray sum;
    Point a = random(busters) //任选一个敢死队员
    calcAng()->sum;  //计算其他所有点到这点的极角保存到sum
    sort(sum);
    cnt1 = 1;
    cnt2 = 0;
    for i: 1 to 2n do
        cnt1 += (sum[i] is buster)
        cnt2 += (sum[i] is ghost)
        if (cnt1 == cnt2 && sum[i] is ghost) break;
    return a->sum[i]; //点a到sum[i]的连线即为所求
```

7.2

```
while (totalPoint > 0)
    line = getLine(busters, ghosts) //用7.1中的算法找到一条线来划分
    deletePoint(); //删除这两个点
```

算法正确性证明

由于没有三点共线，并且数组中只有两中点，我们选择出一个buster之后，剩下序列中的总存在一个以ghost结尾的两中点个数相同的序列。所以此算法可行。

算法复杂度分析

计算极角为$O(N)$，极角排序使用快速排序复杂度$O(nlogn)$，遍历过程$O(N)$，故算法7.1复杂度为$O(nlogn)$，算法7.2仅仅多了一个$O(N)$的选择点的过程，所以复杂度为$O(n^2logn)$。

Assignment 2

2.1

题目

1 Largest Divisible Subset

Given a set of distinct positive integers, find the largest subset such that every pair (S_i, S_j) of elements in this subset satisfies:$S_i\%S_j = 0$ or $S_j\%S_i = 0$.

解答

状态转移方程

先排序，得到序列L，ans[i]表示[0..i]这个区间上的答案集合的大小

初始条件$ans[0] = 1, ans[i] = 0(i > 0)$

$ans[i] = max(ans[i], ans[i]\%ans[j] == 0?ans[j] + 1 : 0)(0 <= j < i)$

算法描述以及正确性证明

满足要求的集合中一定是从小到大并且有着倍数关系。算法每次找到本身之前的所有数字中可以整除自身的数，并取可以得到集合最大值的前驱作为答案。由于排序了所以前面都是比自己小的数字，而自身之前的数字可行的集合已经查找完毕，从而可以从子问题中得到自己的解。如果需要输出集合加一个前驱结点标记即可。

算法复杂度

中间使用了一个二层循环，所以复杂度为$O(N^2)$

2.2

题目

2 Money robbing

A robber is planning to rob houses along a street. Each house has a certain amount of money stashed, the only constraint stopping you from robbing each of them is that adjacent houses have security system connected and it will automatically contact the police if two adjacent houses were broken into on the same night.

1. Given a list of non-negative integers representing the amount of money of each house, determine the maximum amount of money you can rob tonight without alerting the police.
2. What if all houses are arranged in a circle?

解答

2.2.1

状态转移方程

$money[1] = nums[1]$

$money[i] = max(money[i-2] + nums[i], money[i-1])$

$money[i]$ 表示从1到i之间的房间里最多可以抢劫多少。

算法描述以及正确性

emmmm不知道怎么说，因为不能抢相邻的，那就隔一个房子来更新。

复杂度

$O(N)$

2.2.2

在第一问的基础上，1和n变成相邻的，所以最优解在$[1..n-1]$和$[2..n]$之间求解第一问取最大值。

2.3

题目

3 Partition

Given a string s, partition s such that every substring of the partition is a palindrome. Return the minimum cuts needed for a palindrome partitioning of s.

For example, given s = "aab", return 1 since the palindrome partitioning ["aa", "b"] could be produced using 1 cut.

解答

状态转移方程

此题需要两次DP，第一次标记出序列的全部回文串，这里需要枚举长度再枚举起点，如下

$flag[i][i] = true$

$flag[i][j] = (s[i] == s[j] \&\& flag[i+1][j-1])$

然后寻找最优切割方式，已知回文串位置的情况下，就变成了最少线段覆盖问题，状态方程如下：

倒序搜索，初始dp[i]=INT_MAX

$dp[i] = min(dp[i], flag[i][j-1] ? dp[j]+1 : 0) \quad j \in [i+1..n]$

算法描述及正确性证明

算法首先使用dp求得所有的回文串位置，然后将这些回文串当成线段，则目标问题就成了经典的线段覆盖问题，同样有着类似的状态转移方程。

算法复杂度分析

使用了两次$O(N^2)$的DP，复杂度$O(N^2)$

2.4

题目

4 Decoding

A message containing letters from A-Z is being encoded to numbers using the following mapping:

$$\begin{aligned} &\text{A} : 1 \\ &\text{B} : 2 \\ &\cdots \\ &\text{Z} : 26 \end{aligned}$$

Given an encoded message containing digits, determine the total number of ways to decode it.

For example, given encoded message “12”, it could be decoded as “AB” (1 2) or “L” (12). The number of ways decoding “12” is 2.

解答

状态转移方程

$dp[k] = dp[k-1] + dp[k-2] \, , s[k-1..k] <= 26$

$dp[k] = dp[k-1] \, , s[k-1..k] > 26$

算法描述以及正确性

如果都是个位数字，只有一种分割方式，多种方式都是由于那些两位数造成的，所以对于一个长度为k的串，组合方式和k-1和k-2有关，也就是最后两个数作为一个字母还是两个字母，最后累加即可，注意要保证最后两个数字在26以内才可以累加。

算法复杂度

一遍扫描$O(N)$

2.5

题目

5 Longest Consecutive Subsequence

You are given a sequence L and an integer k, your task is to find the longest consecutive subsequence the sum of which is the multiple of k.

解答

可以不使用DP

算法描述

维护一个前缀和$sum[i]$和一个当前前缀和对k取余的结果$tmp = s[i] \bmod k$，再对tmp维护一个位置信息$pos[tmp]$，表示第一次取余得到tmp时的位置，根据同余定理，这段序列的和可以被k整除，线性扫描之后取得最大值即可。

伪代码

```
int ans = 0;
s[0] = 0;
int maxL = 0;
for (int i = 1; i <= n; i++)
{
    s[i] = s[i-1] + L[i];
    tmp = s[i]%k;
    if (pos[tmp] != 0 && i - pos[tmp] > maxL)
    {
        maxL = i - pos[tmp];
    }
    else
    {
        if(pos[tmp] ==0) pos[tmp] = i;  //pos[tmp]实际上只保存tmp第一次出现的位置，从而取得的序列最长
    }
}
```

复杂度

只做了一次线性扫描，复杂度为$O(N)$

Assignment 3

3.1

题目

1 Greedy Algorithm

Given a list of n natural numbers d_1, d_2,...,d_n, show how to decide in polynomial time whether there exists an undirected graph $G = (V, E)$ whose node degrees are precisely the numbers $d_1, d_2, \cdots, d_n$. G should not contain multiple edges between the same pair of nodes, or " loop" edges with both endpoints equal to the same node.

解答

问题描述

问题为节点度分配问题，每一轮迭代将节点的度**降序**排序，然后从度大的一端开始，将自己的边安排在度和自己相邻的节点上，这样分配完成之后，即可验证方案的合法性。

伪代码

```
    cnt = 0; // 表示当前已经有多少节点需要和当前节点分配边
    flag = true;
    for (int i = 0; i < n ; i++)
    {
        sort(d, d + n,less<int>);
        if (d[0] == 0) break;
        for(int j = 1; j <= d[0];j++)
        {
            d[j]-=1;
            if (d[j]<0) //分配失败
            {
                flag = false;
                break;
            }
        }
        d[0] = 0;
    }
answer = flag;
```

正确性证明

- 当方案合法时，本方法的结果一定是正确的
- 当方案不合法时，本方法判断为不合法的情况是当中间节点的度为负数或者分配完成后最后一个节点度不为0，由于已经排序，可以证明这两种情况下的方案一定是不合法的。

综上两种情况可以判断算法的正确性。

复杂度分析

算法经过N次排序，总复杂度为$O(N^2logN)$

3.2

题目

2 Greedy Algorithm

There are n distinct jobs, labeled $J_1, J_2, \cdots, J_n$, which can be performed completely independently of one another. Each jop consists of two stages: first it needs to be *preprocessed* on the supercomputer, and then it needs to be *finished* on one of the PCs. Let's say that job J_i needs p_i seconds of time on the supercomputer, followed by f_i seconds of time on a PC. Since there are at least n PCs available on the premises, the finishing of the jobs can be performed on PCs at the same time. However, the supercomputer can only work on a single job a time without any interruption. For every job, as soon as the preprocessing is done on the supercomputer, it can be handed off to a PC for finishing.

Let's say that a *schedule* is an ordering of the jobs for the supercomputer, and the *completion time* of the schedule is the earlist time at which all jobs have finished processing on the PCs. Give a polynomial-time algorithm that finds a schedule with as small a completion time as possible.

解答

任务分为两个阶段，第二阶段可以并行，直觉上优先在超算上时间短而在PC上时间长的，来获得PC的最大并行度。所有任务在超算上是串行的，所以没有优可言，最终的任务时间取决于PC上的任务安排，所以应该让PC上需要时间更长的任务先运行，所以按照PC上的任务执行时间从大到小排序即可。

复杂度

排序，$O(NlogN)$

3.3

题目

3 Greedy Algorithm

Given two strings s and t, check if s is subsequence of t?

A subsequence of a string is a new string which is formed from the original string by deleting some (can be none) of the characters without disturbing the relative positions of the remaining characters. (ie, "ace" is a subsequence of "abcde" while "aec" is not).

解答

问题描述

只要找到一个顺序正确的子序列就可以，$O(N)$扫描一遍即可。

伪代码

```
    string s,t;
    curr_pos = 0; // 当前匹配的字符在s中的位置
```

```
    flag = true;
    for (int i = 0; i < t.length(); i++)
    {
        if (t[i] == s[curr_pos])
        {
            curr_pos++;
            if (curr_pos == s.length())
            {
                break;
            }
        }
    }
    if (curr_pos < s.length()) flag = false;
    answer = flag;
```

正确性证明

算法执行的过程就是将s串离散在t串里的过程，并且遵守首次匹配的规则，不存在原本有解但是无法找到的问题，同时可以保证找到的解一定是正确的解，所以可以保证算法结果正确。

复杂度分析

单遍扫描，复杂度$O(N)$

3.4

题目

4 Greey Algorithm

Suppose you are given two sets A and B, each containing n positive integers. You can choose to reorder each set however you like. After reordering, let a_i be the ith element of set A, and let b_i be the ith element of set B. You then receive a payoff of $\prod_{i=1}^{n} a_i^{b_i}$. Give an polynomial-time algorithm that will maximize your payoff.

解答

都从小到大排序，越大的数让其幂次越高，总的收益最大。

复杂度

排序 $O(NlogN)$

证明

反证法，在这个排序的方式上做任何改动，都会使目标值变小。

Assignment 4

略

Assignment 5

5.1

题目

1 Load balance

You have some different computers and jobs. For each job, it can only be done on one of two specified computers. The load of a computer is the number of jobs which have been done on the computer. Give the number of jobs and two computer ID for each job. You task is to minimize the max load.

(hint: binary search)

解答

分析

建图，在每个任务和其可以运行的机器建立一个容量为 1 的边，然后添加两个虚拟点s,t，分别连接所有的任务和所有的机器，s 到所有任务的边容量始终为 1 ，对 t 和机器之间的容量进行二分查找，查找边界为$[1,n]$. 求解这个图上$s-t$的最大流，如果最大流量等于任务数量，说明任务全部完成，在这个条件的限制下查找到t和机器之间的容量的最小值，就是负载最大的机器的最小任务数量。

伪代码

```
//加边 然后添加源点汇点
  int l = 1, r = n;
  int s = m + n + 1;
  int t = n + m + 2;
  while (l < r)
  {
      int mid = l + r >> 1;
      //建图 s到任务节点容量为1，任务节点到机器节点容量为1，机器节点到汇点t容量为mid
      buildGraph();
      ans = max_flow(s,t);
      if (ans == n)
          r = mid;
      else
          l = mid + 1;
  }
  ans = l;
```

正确性证明

首先，若一个图存在最大流，dinic算法可以正确计算，此问题中最大流就是可以解决的问题数量，所以在保证任务全被完成的前提下，二分查找负载对大的机器的负载下限是可行的。

复杂度分析

使用dinic 算法和二分查找，任务数为n，机器数为m，本题中边数等于2n，点数等于$n+m$，总的复杂度为$O((m+n)^2 2n\log(n))$

5.2

题目

2 Matrix

For a matrix filled with 0 and 1, you know the sum of every row and column. You are asked to give such a matrix which satisfys the conditions.

解答

分析

问题可建模为有限制的最大流问题，和第一题相同，引入两个虚拟点，s和t， 设置M个行节点和N个列节点，源点和每个行节点相连接，容量为行的和，列节点和汇点相连，容量为列的和，问题有解的前提是所有行的和和所有列的和的和相等。然后所有的行节点和所有的列节点之间建边，容量为1，然后求解这个额图上的最大流，如果最大流等于所有行的和的和同时等于所有列的和的和，那么问题有解，将中间的流量矩阵输出即为答案，否则问题误解。

伪代码

```
int s = m + n + 1;
int t = n + m + 2;
//源点到行节点建边
//列节点到汇点
//行列节点之间建边
buildGraph();
 //计算最大流
maxflow = max_flow(s,t);
//检查所有的附加边是否满流，就是检查dinic算法执行后的反向边
int sum = 0;
int matrix[500][500];
 //获取流量矩阵
matrix = get_flow();
 //检查答案是否合法
if (sum_rows == maxflow && sum_colums == maxflow)
{
    ans = matrix;
}
else return No Answer;
```

正确性证明

矩阵上的每个元素都同时被计算到行列的和中，可以看成是一个流量，在没有行列和限制的情况下，矩阵中的数值分布就是网络上的从行到列的一个流，加上行列和的限制之后，添加好源点汇点，我们希望的答案是满流的，所以求解最大流可以得到答案。

复杂度分析

共n+m+2个点，n*m+n+m条边。使用dinic算法，复杂度为$O((n+m)^2(nm+m+n))$

5.3

题目

3 Unique Cut

Let $G = (V, E)$ be a directed graph, with source $s \in V$, sink $t \in V$, and nonnegative edge capacities c_e. Give a polynomial-time algorithm to decide whether G has a unique minimum st cut.

解答

分析

判断最小割的唯一性，由图的割的性质可以知道，割可以将图分为两部分，如果将割边都隔断，两部分依然为连通图。而对于一个图，最大流对应这最小割，最小割中的边都是满流的，也就是在剩余网络中是不连通的，我们使用求解最大流之后的剩余网络，如果去掉割边之后两部分子图分别连通，则割唯一，否则不唯一。

伪代码

```
_G = maxflow(s,t);
//分别从s和t出发遍历剩余图并标记节点,返回被标记节点的个数
n1 = bfs(_G,s);
n2 = bfs(_G,t);
//如果
if (n1 + n2 == V) ans = YES; else ans = No;
```

正确性证明

如果割不唯一，则剩余网络一定会有节点不可达，反之所有节点可达。

复杂度分析

两个BFS和一次最大流 ,低阶的BFS复杂度忽略不计。总复杂度$O(n^2m)$

5.4

题目

4 Problem Reduction

There is a matrix with numbers which means the cost when you walk through this point. you are asked to walk through the matrix from the top left point to the right bottom point and then return to the top left point with the minimal cost. Note that when you walk from the top to the bottom you can just walk to the right or bottom point and when you return, you can just walk to the top or left point. And each point CAN NOT be walked through more than once.

解答

最小费用流，往返等价于单程走两次，每个边只能走一次，所以流设为1，总共需要走两次，所以要添加超级源点和汇点，连接矩阵中的起点终点，并且流量设为2。矩阵中只有右边和下边的相邻点之间建立边，在这个网络上求解最小费用最大流。

复杂度分析

使用SPFA+邻接表增广的最小费用最大流，同时因为本题流量仅仅为2，所以和最短路时间基本相同，所以复杂度为O(VE)

5.5

题目

5 Network Cost

For a network, there is one source and one sink. Every edge is directed and has two value c and a. c means the maximum flow of the adge. a is a coefficient number which means that if the flow of the edge is x, the cost is ax^2.

Design an algorithm to get the Minimum Cost Maximum Flow.

解答

为了让原始的费用流算法继续工作，要把费用和流量之间的非线性关系转换为线性关系，并且要保证转换是等价的。将x^2 转换为流量的一维函数，将容量为n的边拆分成n个容量为1的边，对应的费用要满足$\sum_{i=1}^{k} cost(i) = k^2$，考虑等差数列前n项和是平方的形势，问题变成了设计一个等差数列使得前n项和为n^2，可以得到这个序列是$1, 3, 5, 7, \ldots 2n-1$, 所以将原问题每条边拆成流量数条边，每条新边的费用按照前述等差数列分配（带上α），在新构造的图上计算最小费用最大流即可。

```
build_graph()
{
    for edge(s,t,w)
    {
        for (int i = 1; i <= w; i++)
        {
            addedge(s,t,1,a*(2*i-1));//同时添加反向边，反向边费用取相反数，流量为0
        }
    }
}
ans = min_cost_max_flow(s,t);
```

正确性证明

最小费用最大流涉及到费用和流量，可以证明新构造的图的最大流和原始图相同，费用在问题分析中也做了说明，满足原始问题的函数关系，由于最小费用流不断寻找最小费用的增广路，所以可以保证算法执行的时候，两个点之间有k的流量的情况下，选择的边一定是前k个边，因而两个图等价。

复杂度分析

使用SPFA+邻接表增广最小费用最大流，复杂度为$O(FVE)$,其中$E = \sum w_i$

5.6

题目

6 Maximum Cohesiveness

Given an undirected graph, each edge is assigned one weight, find a subset S of nodes to maximize $e(S)/|S|$, where $e(S)$ denotes the sum of edge weights in S and $|S|$ is the number of nodes in S. Give a polynomial-time algorithm that takes a rational number α and de- termines whether there exists a set S with cohesiveness at least α.

解答

将每条边都看成一个点，记做e_i， 添加超级源点汇点s,t ,由源点s 连接所有的边节点，容量为边的权重，所有的边节点连接自己的两个端点，容量为无穷大，所有的原始节点连接到t， 容量为α 。然后计算最大流，如果最大流量小于所有的原始边权之和，则存在这样一个子集，否则不存在。找到最大的α 可以使用二分查找。

伪代码

```
build_graph()
{
    cnt = 1;
    tot=0;//边权累加
    for edge(u,v,w)
    {
        //nodecnt为节点数，nodecnt+cnt表示这个边节点编号，addedge包含反向边的添加
        addedge(s, nodecnt+cnt,w);
        addedge(nodecnt+cnt,u,inf);
        addedge(nodecnt+cnt,v,inf);
        addedge(u,t,alpha);
        addedge(v,t,alpha);
        cnt++;
        tot+=w;
    }
}
ans = max_flow(s,t);
if (tot > ans) return true;
else return false;
```

正确性证明

总流量小于边权值说明有边节点的边流se没有满流，而如果一个边没有满流，则他的两个节点和t之间的边一定满流。不存在一个边节点对应的两个原始节点和t连接的边不同时满流的情况。考虑特殊情况，仅有一个se的边没有满流，设这个边节点为e，对应的两个原始节点为v_1,v_2，如果se没有满流，则v_1t,v_2t一定满流，如果这些流量并不是来自于se，而是来自和v_1,v_2连接的其他边，因为和t连接的所有边都等于α, 此时和v_1,v_2连接的其他节点都刚好满流（因为假设只有一个se没有满流）。所以v_1,v_2连接的其他节点所连接的其他原始节点到达t的边都是满流的，这样始终可以找到一个集合S，设其中的点个数为k，流量$k\alpha < \sum_{e\in S} w_{se}$， 这便是我们要找的集合。

5.7

题目

7 Maximum flow

Another way to formulate the maximum-flow problem as a linear program is via flow decomposition. Suppose we consider all (exponentially many) s-t paths p in the network G, and let f_p be the amount of flow on path p. Then maximum flow says to find

$$\begin{array}{rrcl} \max & z & = & \sum f_p \\ s.t. & \sum_{e \in p} f_p & \leq & u_e, \quad for \quad all \quad edge \quad e \\ & f_p & \geq & 0 \end{array}$$

(The first constraint says that the total flow on all paths through e must be less than ue.) Take the dual of this linear program and give an English explanation of the objective and constraints.

解答

The dual of this program has the form as follows:

$$\begin{aligned} & \min \sum u_e l_e \\ s.t. \quad & \sum_{e \in p} l_e \geq 1 \quad for\, all\, path\, p \end{aligned}$$

We add a 'weight' which is called l_e for every edge.

The two constraints says every path has a non-negative weight and sum of every path's weight must be at least 1. Actually, given a cut A-B(and $s \in A, t \in B$) , if edge e connect A and B(in other words, $u \in A, v \in B \quad (u, v)$ is two vertices of edge e， and (u, v) is a cut edge),we setl_e = 1, otherwise, we set l_e=0.

So, the objective function is just to minimize the sum of capacity multiply its weight. Obviously, it is designed to solve the minimum cut problem.