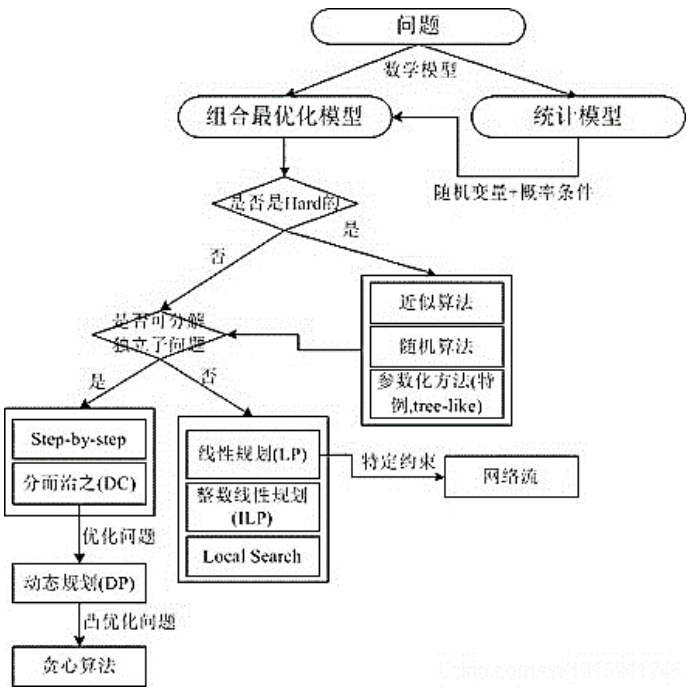
Hints of Assignment 1-7

Algorithm Design and Analysis

Datumor[1], 2015



[1] niujinghao@outlook.com

Assignment 1

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.

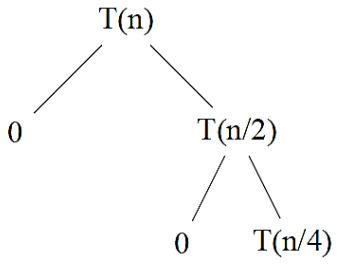
   However, the only way you can access these values is through queries to the databases. In a single query, you can specify a value k to one of the two databases, and the chosen database1 will return the k^{th} smallest value that it contains. Since queries are expensive, you would like to compute the median using as few queries as possible.Give an algorithm that finds the median value using at most O(log n) queries.

Solution: (a)In this issue, the query is used to get the median of one specific set.With the idea of *Divide and Conquer*, we can design an algorithm whose inputs are two databases D1 and D2 and two pointers p1 and p2 which are both initialed as n/2 in the subset, which iteratively finds the medians of the subsets generated by pointers until one step where the set input is not dividable anymore.For each step, the medians we get are called m1(position of p1) and m2(position of p2), we compare the value of them.For example, for the i^{th} step, if $m1 > m2$, then we renew p1 to $p1 - n/2^i$ and p2 to $p2 + n/2^i$, or, we renew them inversely.Thus the real median we want is between p1 and p2.When $i = log_2 n$, we compare the final m1 and m2, return the smaller one, which is the median of joint database.

MEDIAN FOR TWO DATABASES

```
1   p1 = p2 = n/2
2   for i = 2 to log_2 n
3       do m1 = query(D1,p1)
4          m2 = query(D2,P2)
5          if m1 > m2
6             then p1 − n/2^i → p1
7                  p2 + n/2^i → p2
8             else
9                  p1 + n/2^i → p1
10                 p2 − n/2^i → p2
11  return min(m1,m2)
```

(b)Subproblem Reduction Graph:



(c)Prove the Correctness: With the technique of loop-invariant, we can prove it in following steps.

- Initialization: At first step, p1 = p2 = n/2 , they are both the median of their own subset.
- Maintenance:Suppose $m1 > m2$, since m1 is the median of D1, the global median should in the smaller binary part of D1 or bigger part of D2.Then searching field is updated to D1' and D2', while p1 and p2 is also changed to their boundary. During the **for** loop, loop invariant should hold and the algorithm is still searching in potential field.
- Termination: At termination, i = $log_2 n$, m1 and m2 is the $n^{th}$and $n + 1^{th}$smallest point.According to the definition of median, we choose the smaller one of m1 and m2 as the final median.

(d)Analyse the Complexity:for every step,there are two operations need to take, T(n) = 0+T(n/2)+2c,according to Master Theorem,T(n)=$O(logn)$.

2 Divide ans Conquer

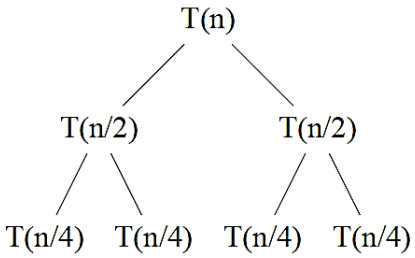
A group of n Ghostbusters is battling n ghosts. Each Ghostbuster is armed with a proton pack, which shoots a stream at a ghost, eradicating it. A stream goes in a straight line and terminates when it hits the ghost. The Ghostbusters decide upon the following strategy. They will pair off with the ghosts, 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. Argue 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.

Q1: (a)We find the bottom, left-most point P0 first, which is assumed one Ghostbuster here.Then we sort the remained points according to the angle of their connection line and basic line.Points in SortedSet are checked in order and when the difference between visited points number of Ghostbusters and Ghost equal to 1, end the loop and return the connection of present point and P0.

```
FINDING PASSING LINE
1  Find the bottom, left-most point P0(Supposed Ghostbusters)
2  Calculate the angle of other points to P0 and sort them to SortedSet
3  Initial FLAG = 0 N_gb = N_gh = 0  //Number of visited Ghostbusters and Ghost
4  for i = 1 to SortedSet.length
5       do FLAG ← N_gb − N_gh
6          if FLAG == 1
7            then
8                 return Line Pi to P0 and end loop
9            else
10                N_gb ← N_gb + (Pi is Ghostbuster)
11                N_gh ← N_gh + (Pi is Ghost)
```

(b)Subproblem Reduction Graph:



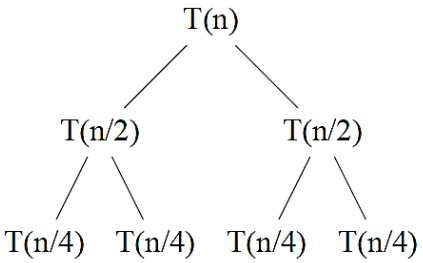
(c)Prove the Correctness: At first step, angle of every points in new set is computed.After we Sort the points with MergeSort Algorithm, points are visited in order and their number are recorded. Once we find the situation suitable, loop is terminated.Because no three positions are collinear, the line generated satisfies the problem's request.

(d)Analyse the Complexity: Besides the MergeSort for n-1 points, the algorithm calculates angles first with cost of c(n-1), which is small compared to MergeSort.Thus in this issue, T(n)=O(nlogn).

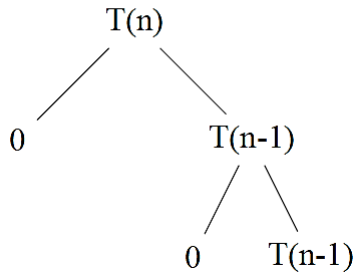
Q2: (a)With the algorithm in Q1, which we may call **FPL(Finding Passing Line)** here, we can find one points pair first.There connection line will divide the plane into two parts, which are the subproblems in next step. Then recursively apply FPL to find points pairs and their subproblem until one side of the line contains no Ghostbusters or Ghosts.

```
FINDPAIR(LEFT-SET,RIGHT-SET)
1  FPL(left-Set)
2  FPL(right-Set)
3  if There are points in left-Set and right-set
4     then FindPair(left-Set,right-set)
```

(b)Subproblem Reduction Graph:
Best Case:



Worst Case:



(c)Prove the Correctness: At first step, we get the initial input, one point pair and the subsets it generates.FPL finds the point pair of each subset.Then, the algorithm recursively find the new points pair until the subset is empty.The loop invariant holds initially and is maintained during the recursive process.
(d)Analyse the Complexity:the complexity of this algorithm is determinded by the time we take on sorting,thus the worst case is when one of the subset is always empty.In this case, we need n/2 total iterations to find pairings,$T(n) = O(n^2 log_n)$.

3 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.

1) Describe in natural language:
   Same idea as merge sort with counting inversions,but this time we use an extra for-loop to count besides a for loop for sorting.

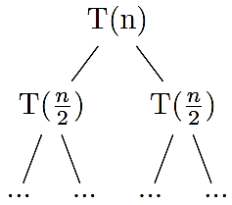
PSEUDO-CODE:

```
function SIG_INVERSION(A)
    if Length(A) ≤ 1 then
        return [0,A]
    n ← Length(A)/2
    L, A_L ←SIG_MERGE(A[1, n])
    R, A_R ←SIG_MERGE(A[n + 1, Length(A)])
    N, A_C ←COMBINE_WITH_COUNT(A_L, A_R)
    return [L + R + N, A_C]

function COMBINE_WITH_COUNT(A, B)
    i ← 0, j ← 0, count ← 0
    C ← []
    for p = 1 : Legth(A) + Length(B) do
        if i > Length(A) then
            C ← [C, B[j]]
            j ← j + 1
            Continue
        if j > Length(B) then
            C ← [C, A[i]]
            i ← i + 1
            Continue
        if A[i] ≤ B[j] then
            C ← [C, A[i]]
            i ← i + 1
        else
            C ← [C, B[j]]
            j ← j + 1
    i ← 0, j ← 0
    for p = 1 : Legth(A) + Length(B) do
        if i > Length(A) then

            Break
        if j > Length(B) then
            Break
        if A[i] > 3 * B[j] then
            count ← count + (Length(A) + 1 − i)
            j ← j + 1
        else
            i ← i + 1
    return [count, C]
```

(b)Tree:



(c)Proof:

1) The Base case is apparently correct.

2) Assume each subprogram is correct:
Since the 2 arrays are sorted,when we have $A[i] > 3 * B[j]$,we have $A[i, Length(A)] > 3 * B[j]$.So we prove it.

(d)Complexity:

1) Time:
It's easy to know that it's the same as mergesort $O(n \log n)$

2) Space:
It's easy to know that it's the same as mergesort $O(n)$

下面是解法2：

算法描述：根据题目的描述，本题可以按照一般求逆序数的思路来，除了在计算逆序数加一的时候加入$a_i > 3a_j$判断即可。根据定义，我们这样定义一个序列的逆序数: 序列$a_1,a_2,a_3 \dots a_n$，这个序列的逆序数 C, 等于a_1,a_2 ...的逆序数的和.即 C=sum(C_i)，C_i为满足$a_i > a_j$ (j > i)的数的总的个数，即C_i= sum($a_i > a_j$) (j >i)。我们一般写的算法一般会做 N(N-1)/2 次比较，时间复杂度为: O(N^2)。
下面采用的分而治之的思想来改进，算法思想如下：
假设我们将序列$a_1,a_2,a_3 \dots a_n$分成两份: B_0=(a_1 a_{12} $a_{n/2}$) B_1= ($a_{\frac{n}{2}+1} \dots a_n$)，那么 C=C($B_0$)+C($B_1$)+M($B_0B_1$)。

Pseudo-code:

```
SIGNIFICANT-INVERSIONS(L,star,end)
1: n ← L.length //数组 L 的长度
2: if n=1
3:     return 0
4: else
5:     p= ⌊(star + end)/2⌋ //得到B0和B1
6:     N1 ← SIGNIFICANT-INVERSIONS(L,star,p)
7:     N2 ← SIGNIFICANT-INVERSIONS(L,p+1,end)
8:     N3 ← MERGE (L,star,p,end)
9: return N = N1 + N2 + N3
```

```
Merge(L,star,p,end)
 1: L←L[star…p]
 2: R←L[p…end]
 3: i ←1
 4: j ←1,
 5: InversionCount ← 0
 6: for k=p to end
 7:     if L[i] > R[j]
 6:         A[k] =L[i]
 7:         i++
 8:     if L[i] > 3R[j] then
 9:         InversionCount = InversionCount+j-k
10:     else
11:         A[k]=R[j]
12:         j++
13: end for
```

算法正确的分析：根据算法的描述可以自证算法的正确性。

算法的复杂度：$T(n) \leq 2T\left(\frac{n}{2}\right) + cn$ 其中 Merge(L,star,p,end)的复杂度为 cn，又因为每一次递归，规模减半，所以最终的复杂度为$O(nlogn)$。

4 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.

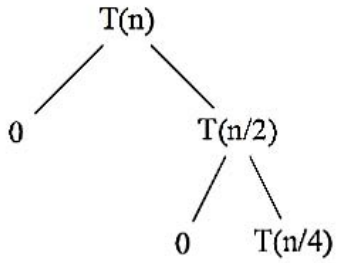
You are given such a complete binary tree T, but the labeling is only specified in the following *implicit* way: for each node v, you can determine the value x_v by *probing* the node v. Show how to find a local minimum of T using only $O(\log n)$ *probes* to the nodes of T.

Solution: (a)In this problem, we define probe() to get the value of one node.We recursively find the smallest one in its children and itself.If itself has the smallest value, then we return its value or we recursively compare the smallest child and its subtree.

-Pseudo-code:

```
FINDMIN(T)
1  if min(probe(T.left()), probe(T.left()), probe(T.root())) == probe(T.root())
2     then
3          return T.root()
4     else
5          if probe(T.left()) > probe(T.right())
6            then
7                 FindMin(T.right())
8            else
9                 FindMin(T.left())
```

(b)Subproblem Reduction Graph:



(c)Prove the Correctness: With the technique of loop-invariant, we can prove it in following steps.

- Initialization: At first step, the initial node we input is the root, we compare the value of it and its two children. The node we may return is the smallest of them.
- Maintenance:Suppose the right child is smallest, we compare the values of this new node and its children, this process recursively goes forward until one node whose root has the smallest value.
- Termination: At termination, since the root's value is the smallest, the local minimum is found and it is the smallest in the path through.

(d)Analyse the Complexity: In a worst-case, T(n)=T(n/2)+cn , where the operation path goes through from root to the final edge. The number of layer for the Tree is equal to the times we use probes. According to Master Theorem,T(n)=$O(logn)$.

5 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 4**. 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:考查这六条直线$i = 1; i = n; j = 1; j = n; i = \lfloor (1 + n)/2 \rfloor; j = \lfloor (1 + n)/2 \rfloor$. 上的点的值，并找出最小值对应的点。如果这个最小点的邻点的值比这个点的值大，那么直接返回这个最小值点，它就是局部最小值点。否则，因为这六条直线将网格分成了四个部分，最小值点所在的部分中一定含有极小值点。因此只需要在这个区域递归地求解。

注：满足O(n) 的原因是T(n) = 1*T(n/2)+O(n)，后边的f(n) = O(n)是每步求最小值算法的复杂度。

6 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.

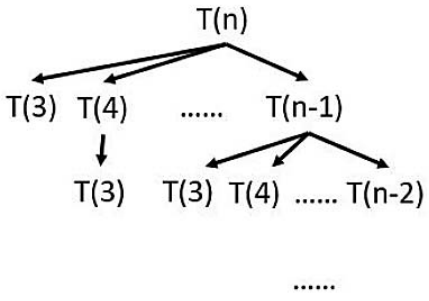
Give an algorithm that decides how many ways we can divide a convex polygon with n vertices into triangles.

6.1 算法思路

对于一个凸多边形有 n 个顶点，要把它划分成多个三角形，用一条线连接不相邻的两个定点后，被划分为两个多边形，假设 f(n)代表有 f(n)种划法，划分后的多边形一个有 k 个顶点，有 f(k)种划法，则另外一个有 n+2-k 个顶点，有 f(n+2-k)种划法，因此得到递推公式：f(n) = f(3)*f(n+2-3) + ...+f(n+2-3)*f(3), n>2，f(3)=1.

```
Int f(int n)
    If n==3
        Return 1;
    Else
        Int count=0;
        For i=3 to (n-1)
            Count += f(i)*f(n+2-i);
            Save every f(i) in a global array
        Endfor
        Return count
    Endif
```

6.3 子问题图



如上的子问题图，递归树有 n-2 层，由于使用全局数组保存每一个 f(i)，所以每个 f(i)只需计算一次即可， T(3)时间复杂度是 O(1)，在上一层已计算过的也是 O(1)，所以从第 1 层到第 n-2 层，时间复杂度将达到 $O(1+n-2+...+1) = O(n^2)$。如果不保存 f(i)，导致每层重复计算，则时间复杂度将达到 $O(2^n)$。

注：上边这个做法有点问题仅供参考，实际上是katalan数通项从n-2开始

算法描述：在凸多边形的划分方案中，我们也是采用分而治之的思想，讲一个凸多边形分成两个凸多边形，然后再把每一个凸多边形在分成两个凸多边形，直到化成三角形为止。因为凸多边形的任意一条边必定属于某一个三角形，所以我们以某一条边为基准，以这条边的两个顶点为起点P_1和终点P_n，将该凸多边形的顶点依序标记为$P_1,P_2...P_n$，再在该凸多边形中找任意一个不属于这两个点的顶点P_k（2<=k<=n-1），来构成一个三角形，用这个三角形把一个凸多边形划分成两个凸多边形，其中一个凸多边形，是由$P_1,P_2...P_k$构成的凸 k 边形（顶点数即是边数），另一个凸多边形，是由$P_{k+1},P_{k+2}...P_n$构成的凸 n-k+1 边形。

此时，我们若把P_k视为确定一点，那么根据乘法原理，凸多边形的分解的数量($f_{(n)}$)就等价于——凸 k 多边形的划分方案数乘以凸 n-k+1 多边形的划分方案数，即选择P_k这个顶点的$f_{(n)}=f_{(k)}\times f_{(n-k+1)}$。而 k 可以选 2 到 n-1，所以再根据加法原理，将 k 取不同值的划分方案相加，得到的总方案数为：

$$f_{(n)} = f_{(2)}f_{(n-2+1)} + f_{(3)}f_{(n-3+1)} + \cdots + f_{(n-1)}f_{(2)}\ （n>3）$$

根据公式，我们可以得到算法设计如下：

Pseudo-code:

```
CONVEX_POLYGON(n)
1: sum←0
2: if n<=3
3:     return 1
4: for i=2 to n-1
5:       sum←sum+ CONVEX_POLYGON(i) *CONVEX_POLYGON(n-i+1)
6: return sum
```

算法正确的分析：根据算法的描述可以自证算法的正确性。

算法的复杂度：根据上面的公式，我们会发现一个 n 的问题会被分成 n-1 个子问题，并且每一个子问题包含 n-1 个乘法，所以算法的复杂度为$O(n^2)$。

Assignment 2

1 Money robbing

A robber 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?

Solution:(a)We define that there are n house along this street, P(i) contains the money amount in the i^{th} house.For the i^{th} house, OPT(i) is defined to be the best money amount he gets, when the robber arrives the i^{th} house in this street.In which,he has to decide whether to rob this house.

If he robs this house and he will get OPT(i-2)+P(i),or he will get OPT(i-1),since he can not rob adjacent houses.Thus the general form of **sub-problem** is to get largest amount of money when the robber arrives the i^{th} house.

DP equation: $OPT(i) = max[(OPT(i-2) + P(i)), OPT(i-1)]$

```
OPT-ROB(i)
 1  if (i == 0)
 2    then
 3         return 0;
 4    else
 5         if (i == 1)
 6           then
 7                return P(1);
 8           else
 9                OPT = max((OPT-ROB(i-2)+P(i)), OPT-ROB(i-1));
10                return OPT;
```

Prove the Correctness: Suppose for i_{th} house, there is a better value OPT'(i) better than OPT(i), without loss of generality, OPT(i) chooses to rob at i, then, OPT(i) = OPT(i-2)+P(i).Since OPT'(i) is a different choice, which should be OPT(i-1).What's more, OPT(i-1) should larger than OPT(i-2)+P(i):a contradiction to the definition of OPT function.

Analyse the Complexity:for every recursing step,there is only constant level complexity. Thus,T(n)=$O(n)$.

(b)if all of the houses are arranged in circle, the problem structure is similar.However, because the first and the last house are connected, the robber can not rob both of them.Thus, we could divide the original problem into two subproblem:including the first house and not including the first house or, not including the first house but including the last house.In the end, we can compare the results of this size(n-1) subproblem and return the bigger one.

```
OPT-ROB-CIRCLE-FIRST(i)
 1  if (i == 0)
 2     then
 3          return 0;
 4     else
 5          if (i == 1)
 6            then
 7                 return P(1);
 8            else
 9                 OPT = max((OPT-ROB(i-2)+P(i)), OPT-ROB(i-1));
10                 return OPT;

OPT-ROB-CIRCLE-LAST(i)
 1  if (i == 1)
 2     then
 3          return 0;
 4     else
 5          if (i == 2)
 6            then
 7                 return P(2);
 8            else
 9                 OPT = max((OPT-ROB(i-2)+P(i)), OPT-ROB(i-1));
10                 return OPT;

OPT-ROB-CIRCLE(n)
1  FIRST = OPT-ROB-CIRCLE-FIRST(n − 1);
2  LAST = OPT-ROB-CIRCLE-LAST(N);
3  return max(FIRST, LAST);
```

Prove the Correctness: the new problem is composed of two similar original issues, with one extra comparation, which does not affects the correctness.

Analyse the Complexity: Same as the original problem, for every recursing step,there is only constant level complexity.We have two subproblem with T(n-1) and one extra comparation. Thus,T'(n)=2T(n-1)+1 = $O(n)$.

2 Minimum path sum

Find the minimum path sum of a triangle from top to bottom. Each step you may move to adjacent numbers on the row below.

For example, given the following triangle:

$$\begin{array}{c} 2 \\ 3\ 4 \\ 6\ 5\ 7 \\ 4\ 1\ 8\ 3 \end{array}$$

The minimum path sum from top to bottom is 11 (**i.e.**, $2+3+5+1=11$).

Solution:,If P is the item Matrix, P(i,j) represents the item in $(row_i, column_j)$.The basic idea is to search for the best item from the bottom to the top.We define OPT(i,j) for the biggest sum of path whose end is the item in $(row_i, column_j)$. There are potential items may in this path, P(i-1, j) and P(i-1, j-1). Specially, the item from the edge can only be determined by the item over it.Thus the general form of **sub-problem** is to get smallest path sum when we know the end of the path is at $(row_i, column_j)$.

DP equation: $OPT(i,j) = min[OPT(i-1,j-1), OPT(i-1,j)] + P(i,j)$ Particularly, if $i = j$ OPT(i,j)=OPT(i-1,j-1)+P(i,j); and if $j = 1$ OPT(i,j) = OPT(i-1,1)+P(i,j).

```
OPT-MIN-PATH(i, j)
 1  if (i == 1)
 2     then
 3          return P(1,1);
 4     else
 5          if (i == j)
 6            then
 7                 return OPT-MIN-PATH(i-1,j-1)+P(i,j);
 8            else
 9                 if (i == 1)
10                   then
11                        return OPT-MIN-PATH(i-1,1)+P(i,j);
12                   else
13                        return min[OPT-MIN-PATH(i-1,j-1),OPT-MIN-PATH(i-1,j)]+P(i,j);
```

```
OPT-MIN-FIND(n)
1  min-path = infinite;
2  for j = 1 to n
3       do
4          if (OPT-MIN-PATH(n,i) < min-path)
5            then main-path = OPT-MIN-PATH(n,i);
6  return min-path;
```

Prove the Correctness: Suppose for (i, j) item, there is a better value OPT'(i,j) better than OPT(i,j), without loss of generality, OPT(i,j) is in the path from (i-1,j-1), then, OPT(i,j) = OPT(i-1,j-1)+P(i,j).Since OPT'(i) is a different choice, which should be OPT(i-1,j).What's more, OPT(i-1,j) should larger than OPT(i-1,j-1):a contradiction to the definition of OPT function.

Analyse the Complexity:for every recursing step in OPT-MIN-PATH,there is only constant level complexity.We have n times of OPT-MIN-PATH to compare, Thus,T(n)=$O(n^2)$.

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.

Solution: In this problem, we define the length of sequence equal to n, and the cut[i](i is from 1 to n) in vector cut stores the minimum of cut number from the i_{th} character to the end of the sequence, which we use OPT(i) to discuss in the following explanation.For the worst situation, we have to cut between each pair of character.Thus we have the initial definition of cut: for every element, cut[i] = n - i.Then we create a new matrix Judge(n*n) to store the palindrome information, where Judge[i][j] is True if sequence[i] to sequence[j] is palindrome.For one palindrome(we use j to describe the end index of palindrome), we can choose to take it as a partition and add to cut vector, or do nothing.Thus the general form of **sub-problem** is to get smallest number of cut number for i_{th} character to the end.

DP equation: $OPT(i) = min[OPT(i), OPT(j+1)+1]$

```
PARTITION(S)
 1  min = 0;
 2  n = S.length();
 3  for i = 1 to n
 4       do
 5           cut [i] = n-i;
 6  for i = n-1 to 1;
 7       do
 8           for j = i to n-1;
 9               do
10                   if (S[i] == S[j] && j-i < 2) || (S[i] == S[j] && M[i+1][j-1])
11                   M[i][j] = True;
12                   cut[i] = min(cut[i], cut[j+1]+1);
13  return min = cut[0];  //返回第0到n的最优划分数
```

Prove the Correctness: Suppose for (i) item, there is a better value OPT'(i) = cut'[i] better than OPT(i), without loss of generality, OPT(i) is determined from index i to j.Since OPT'(i) is a different choice, which should be ended with j'.That cut'[i to j'] should smaller than cut[i to j]:a contradiction to the definition of OPT function.

Analyse the Complexity:for every recursing step in cut vector loop,there is O(n) level complexity.We have n-1 times of comparison, Thus,T(n)=$O(n^2)$.

4 Subsequence Counting

Given a string S and a string T, count the number of distinct subsequences of T in S.

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).

Here is an example: S = "rabbbit", T = "rabbit". Return 3.

解法三：dp思想。

```
<code>如果T[i] == S[j]则 dp[i][j] = dp[i][j + 1] + dp[i + 1][j + 1]
如果T[i] != S[j]则 dp[i][j] = dp[i][j + 1]
公式简单，程序更简单
</code>
```

```
<code class="language-java hljs ">public class Solution {
    /**
     * @param S, T: Two string.
     * @return: Count the number of distinct subsequences
     */
    public int numDistinct(String S, String T) {
        if (T.length() == 0 || S.length() < T.length())
            return 0;

        int[][] dp = new int[T.length() + 1][S.length() + 1];
        Arrays.fill(dp[T.length()], 1);
        for (int i = T.length() - 1; i >= 0; i--) {
            for (int j = S.length() + i - T.length(); j >= 0; j--) {
                if (S.charAt(j) == T.charAt(i))
                    dp[i][j] = dp[i][j + 1] + dp[i + 1][j + 1];
                else
                    dp[i][j] = dp[i][j + 1];
            }
        }

        return dp[0][0];
    }
}</code>
```

1 算法描述

a) 记 dp[i][j]表示两个字符串 S[0,1,…i]和 T[0,1,…j],满足题目要求的种数。

b) 显然对于 S[i],可以不选，此时 dp[i][j] += dp[i-1][j];如果选的话，要求 S[i]=T[j],此时有 dp[i][j] += dp[i-1][j-1]

2 转移方程

```
dp[i][j] = dp[i-1][j]
if S[i] == T[j]:
    dp[i][j] += dp[i-1][j-1]
```

3 伪代码

```
Cal(S, T):
    For i in range(len(S)):
        For j in range(len(T)):
            dp[i][j] = dp[i-1][j]
            if S[i] == T[j]:
                dp[i][j] += dp[i-1][j-1]
    return dp[len(S)-1][len(T)-1]
```

4 时间复杂度

显然时间复杂度为 O(n^2)

5 Decoding

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

A: 1
B: 2
...
Z: 26

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.

分析：

边界条件：

1. 输入字符串长度为0时，则为0

2. 输入字符串的第一个数不能为0，若为0，则为0

根据例子我们可以知道问题可以分解成sum[i]=sum[i-1]+sum[i-2] (i>1).

可以采用自底向上法的思想，有顺的将子问题有小到大进行求解。

代码如下：

```
public int numDecodings(String s) {
    if(0 == s.length()) return 0;
    if(s.charAt(0)=='0') return 0;
    int []num = new int[s.length()];//记录遍历到字符串第i位置时的状态（该状态指的是编码的方法数）
    num[0] = 1;
    for(int i=1;i<s.length();i++){
      if(s.charAt(i) != '0') num[i] = 1;
      String temp = s.substring(i-1, i+1);
      if(temp.charAt(0)=='0') continue;
      if(Integer.parseInt(temp)>0&&Integer.parseInt(temp)<27){
        if(1==i){
          num[i]+=1;
        }else{
          num[i]+=num[i-2];
        }
       }
      }
    return num[s.length()-1];
  }
```

4 Problem 5

4.1 算法分析

对于一个字符串，2241121，排列组合可能有很多可能性，比如11可以看作‘K’也可以看作2个'A'，12可以看作'J'，也可以看作'A"B'，如果我们从头开始分析，没有什么结果。我是从尾部开始分析。

假设：

$c(str)$ 表示字符串str的组合数。

$d(i)$ 表示字符串第i位到**尾部**的组合数。

$has(str)$ 表示字符串str有没有对应的解码，返回$1/0$

现在来看这个字符串：2241121

令$d(n) = c('') = 1$

$d(n-1) = c('1') = 1$

$d(n-2) = c('21') = has('2') * d(n-1) + has('21') * d(n) = 1*1 + 1*1 = 2$

$d(n-3) = c('121') = has('1') * d(n-2) + has('12') * d(n-1) = 1*2 + 1*1 = 3$

$\dots$

$d(0) = c(s) = has('s[0]') * d(1) + has('s[0]s[1]') * d(2)$

通用公式，当$0 <= i < n-2$时：

$d(i) = c(s[i-n]) = has(s[i]) * d(i+1) + has(s[i]s[i+1]) * d(i+2)$

考虑到我们每次只需要当前字符和当前字符后一个字符，然后需要2个变量来保存之前算过的2个组合数。

再来看一个比较特殊的情况，第i位为‘0’怎么处理？我们知道如果出现‘0’，那么它只能和前面一个数组成10/20，不可能单独出来，所以i和i-1位共同组成一个编码，虽然has('1/2') = 1, has('10/20') = 1，但其实它只有一种组合，不存在1/2单独拿出来编码，把0挂单的情况。所以它的组合数其实就是$d(i+1)$，那这怎么得来呢？

我们知道，has('0') = 0，并且has('0x') = 0（这里x代表0-9）。所以第i位为$-0-$的d(i) = 0+0 =0 !! 这也是能理解的，这说明不存在以0开头的任何组合。

我们再来看$d(i-1) = has('1'/'2') * d(i) + has('10'/'20') * d(i+1) = 1*0 + 1*d(i+1) = d(i+1)$满足我们前面说的情况。同时对于最后一位为0的情况，我们也令d(n) = 0；说明本算法也可以处理第i位为‘0’的情况。

4.2 代码

Code Listing 6: Decoding

```
   int Decoding(string s) {
        int d_i1,d_i2 = 1,d=0;
        int a;
        if(s.length() == 0)
            return 0;
        if(s.length() == 1)
            return s[0] == '0'?0:1;
        //我们从倒数第二位开始运算，所以必须把最后一位的值首先确认作为初始条件
        d_i1 = s[s.length()-1] == '0'?0:1;
        //运算正式开始d(i) =  has(s[i])*d(i+1) + has(s[i]s[i+1])*d(i+2)
        for(int i = s.length()-2;i >= 0;i--)
        {
            if(s[i] == '0')
            {
                d_i2 = d_i1;
                d_i1 = 0;
                d = 0;
                continue;
            }
            //除了第一位为0 的情况，1-9 都有对应的编码has(s[i]) = 1 ，所以直接加 d_i1
            d = d_i1;
            //考虑s[i]+s[i+1] 的两位情况
            a = atoi(s.substr(i,2).c_str());
            if(a <= 26)//如果has = 1
            {
                d += d_i2;
            }
            d_i2 = d_i1;
            d_i1 = d;
        }
        return d;
    }
};
```

4.3 复杂度分析

由于只用遍历一遍，复杂度为$O(n)$

Assignment 3

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.

Solution:(a)For the listS_0 = <d1, d2,...dn>, we firstly do the sorting operation. Then, we get one non-increasing sequence S_0.For each step, we subtract d1 from the sequence and subtract 1 from the left first number of d1 elements.Sort the left elements again. Now the subproblem is that given one non-increasing sequence S_i, we should find whether it satisfies the rule above. Noticed that if any $d_j < 0$ situation occurs, the output judgement should be failure.If all the elements in one subsequence are 0, we should return success.

```
GREEDY-GRAPH-JUDGE(S, N)
 1  S = Sort(S);
 2  if (S[1] == 0)
 3     then
 4          return True;
 5          Break;
 6     else
 7          if (S[n] < 0)
 8            then
 9                 return False;
10                 Break;
11            else
12                 k = d1;
13                 n = n-1;
14                 S = S - S[1];
15                 for j=1 to k
16                     do
17                         S[j] = S[j] - 1;
18                         Greedy-Graph-Judge(S, n);
```

Prove the Correctness:

- True Output: If there is one step we get the sorted sequence S, and the biggest element equals to 0 and smallest element > 0, thus all the elements are 0.From the first operation to this step, we connect every subtracted element with those k elements, there is no points left anymore in the end, and the rule is satisfied.
- False Output: If there is at least one point's value < 0, at the subproblem before this step, the element subtracted should have the degree larger than then length of points sequence, which could not find enough pair points to connect, thus False Output is right.

Analyse the Complexity:for every recursing step, the sorting operation holds O(nlogn) complexity.Thus, considered the iteration, at the worst case, we will take O(n) recursing operations,$T(n) = O(n(nlogn + cn)) < O(n^2)$

注：上边最后一个应该是大于号，最终是 O(n^2logn)复杂度，如果只开始排序一次，每步不另外排序，则可得到 O(n^2)的复杂度。

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.

Solution:we firstly sort the jobs sequence to a non-increasing order according to f_i value.Then we take the jobs according to this sequence, from the start point to the end.(Put into the supercomputer and then send to one PC when finished)We define F as the list stored f_i information of J.

Pseudo-code:

MIN-PC-TIME(F)

1 F-sort = sort(F); %with a non-increasing order
2 **return** J_i in the order of F-sort

Prove the Correctness: The subproblem is to decide the order of every two adjoined jobs.Without losing generality, we define p1 and f1 for Job1, p2 and f2 for Job2.Because the PCs' resource is absolutely sufficient, we should start fi at once when pi finished.Also, we should start another pj at once when pi finished.Then, the overall time to finish J1 and J2 holds 4 possible cases:

(a) if p1 is in front of p2 and f1>p2: p1 +f1
(b) if p1 is in front of p2 and f1<p2: p1 +f1 +f2
(c) if p2 is in front of p1 and f1>p2: p1 +f1 +p2
(d) if p2 is in front of p1 and f1<p2: p1 +f1 +f1

Without losing generality, we define f1 > f2, then there is only two possible cases:(a) and (d).Because time(d) - time(a) = p2 > 0, we should chose case(a).Thus we should have a schedule according to a non-increasing order of Job.f.

Analyse the Complexity: With an ideal sorting algorithm, the complexity is O(nlogn).Thus, T(n)= cn + O(nlogn) = O(nlogn).

注：上边那个(a) 有点小问题，其实就是固定首尾相接的 p1p2 比较可变的 p_i+f_i 与 p_j 的长度关系，读者自己验证一下吧。

3 Greedy Algorithm

In a party, a host ask n boys and n girls to play a game, in which a boy and a girl have to walk towards the finish line while keeping a balloon staying between their heads. This game needs cooperation, but height difference between the two players is also important. Suppose the height of boys are $b_1, b_2, \cdots, b_n$ and the height of girls are $g_1, g_2, \cdots, g_n$. The problem is to match n girls and n boys such that the *average difference* between the corresponding players is minimized. That is, you want to minimize $\frac{1}{n}\sum_{i=1}^{n}|b_i - g_i|$. Give a polynomial-time algorithm to solve this problem.

The input consists of n skiers with heights p_1, p_2,...,p_n, and n skies with height s_1, s_2,...,s_n. The problem is to assign each skier a ski to minimize the **AVERAGE DIFFERENCE** between the height of a skier and his/her assigned ski. That is, if the skier i is given the ski a_i, then you want to minimize:

符号修改：

$\Sigma_{i=1}^{n}(|p_i - s_{a_i}|)/n$

Solution:

The key observation is that there is no advantage to "cross matching" reversing the height if order of skiers and skis. That is if $s_1 < s_2$ and $p_1 < p_2$ there is no reason to match s_1 with p_2 and s_2 with p_1.To show this, we have to look at all the possible relationships of the 4 heights and show that the "cost" if matching the shorter of the skiers with the shorter of the skis is always at least as good as the cost of cross matching them. Without loss of generality we can assume that the height of s_1 is the smallest of the 4 heights. With this assumption, we have 3 cases (i.e.: the 3 possible orderings of s_2, p_1, and p_2 given that $p_1 < p_2$).

- $s_1 < s_2 < p_1 < p_2$
  $|p_1 - s_1| + |p_2 - s_2| = p_1 - s_1 + p_2 - s_2 = |p_1 - s_2| + |p_2 - s_1|$.
- $s_1 < p_1 < s_2 < p_2$
  $|p_1 - s_1| + |p_2 - s_2| = p_1 - s_1 + p_2 - s_2 < |p_1 - s_2| + |p_2 - s_1| = s_2 - p_1 + p_2 - s_1$.
- $s_1 < p_1 < p_2 < s_2$
  $|p_1 - s_1| + |p_2 - s_2| = p_1 - s_1 + p_2 - s_2 < |p_1 - s_2| + |p_2 - s_1| = s_2 - p_1 + p_2 - s_1$.

To show that there is always an optimal solution with no cross matching, let S be an optimal matching. If S has no cross matches, we are done. Otherwise, consider two skiers and two skies in a cross match, and reverse their matching so that the shorter of the two skiers has the shorter of the two skis. Using the cases above one can show that this change cannot increase the cost of the match, to the revised solution is at least as good as S.

Simply sort both the skiers and the skis by height, and match the i-th skier with the i-th pair of skis. This algorithm is optimal because another assignment would have a cross match, hence could not have a better cost. The running time of the algorithms is just the time it takes to sort the two list, $O(n \log n)$.

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.

4.1 Description

The a_i and b_i should be ordered consistently. That is, if $a_i < a_j$, $i \neq j$, then $b_i < b_j$. So we can sort a_i and b_i both in increasing order or both in decreasing order to solve the problem.

The following pseudo-code shows the details to solve the problem. It is so simple that doesn't need explanation.

```
REORDERING(A, B)
1  sort A in increasing order of a_i
2  sort B in increasing order of b_i
```

4.2 Proof

We should prove that in the order of the maximum payoff, for each $a_i < a_j$, $i \neq j$, we have $b_i < b_j$. Suppose there exist some $i \neq j$ such that $a_i < a_j$ and $b_i > b_j$. We can swap b_i and b_j to achieve more payoff.

$$a_i^{b_j} a_j^{b_i} = a_i^{b_j} a_j^{b_i - b_j} a_j^{b_j} > a_i^{b_j} a_i^{b_i - b_j} a_j^{b_j} = a_i^{b_i} a_j^{b_j}$$

The conclusion appears to contradict assumptions. So there doesn't exist any $i \neq j$ such that $a_i < a_j$ and $b_i > b_j$.

4.3 Analysis

The algorithm takes two sorting steps. Hence, the complexity is $O(n \log n)$.

Assignment 4

1 Linear-inequality feasibility

Given a set of m linear inequalities on n variables $x_1, x_2, \cdots, x_n$, the **linear-inequality feasibility problem** asks if there is a setting of the variables that simultaneously satisfies each of the inequalities.

1. Show that if we have an algorithm for linear programming, we can use it to solve the linear-inequality feasibility problem. The number of variables and constraints that you use in the linear-programming problem should be polynomial in n and m.

2. Show that if we have an algorithm for the linear-inequality feasibility problem, we can use it to solve a linear-programming problem. The number of variables and linear inequalities that you use in the linear-inequality feasibility problem should be polynomial in n and m, the number of variables and constraints in the linear programming.

- METHOD 1 Set the object function to be a constant number, say, 0.
- METHOD 2 Without losing generality, suppose the original linear-inequality feasibility problem (LIFP) is given like this

$$\begin{array}{rcl} \sum_j a_{ij} x_j & \leqslant & b_j \\ x_j & \geqslant & 0 \end{array}$$

, which has m inequalities in total. We can add a variable x_0 and construct an LP like this

$$\begin{array}{lrcl} \max & -x_0 & & \\ \text{s.t.} & \sum_j a_{ij} x_j - x_0 & \leqslant & b_j \\ & x_j & \geqslant & 0 \\ & x_0 & \geqslant & 0 \end{array}$$

2. There are at least two methods to solve this problem. They are shown as follows.

   - METHOD 1 Without losing generality, suppose the LP is given like this:

$$\begin{aligned} \min \quad & c^T x \\ s.t. \quad & Ax \leqslant b \\ & x \geqslant 0 \end{aligned}$$

   Then its duality is

$$\begin{aligned} \max \quad & y^T b \\ s.t. \quad & y \leqslant 0 \\ & y^T A \leqslant c^T \end{aligned}$$

   From strong duality, we know that optimal values of the two object functions are the same. In other words, $c^T x \geqslant OPT \geqslant y^T b$. As a result, we can write the following linear-inequality feasible problem, which can be settled with the algorithm for linear-inequality feasible problem.

$$\begin{aligned} Ax &\leqslant b \\ x &\geqslant 0 \\ y &\leqslant 0 \\ y^T A &\leqslant c^T \\ c^T x &= y^T b \end{aligned}$$

2 Airplane Landing Problem

With human lives at stake, an air traffic controller has to schedule the airplanes that are landing at an airport in order to avoid airplane collision. Each airplane i has a time window $[s_i, t_i]$ during which it can safely land. You must compute the exact time of landing for each airplane that respects these time windows. Furthermore, the airplane landings should be stretched out as much as possible so that the minimum time gap between successive landings is as large as possible.

For example, if the time window of landing three airplanes are [10:00-11:00], [11:20-11:40], [12:00-12:20], and they land at 10:00, 11:20, 12:20 respectively, then the smallest gap is 60 minutes, which occurs between the last two airplanes.

Given n time windows, denoted as $[s_1, t_1], [s_2, t_2], \cdots, [s_n, t_n]$ satisfying $s_1 < t_1 < s_2 < t_2 < \cdots < s_n < t_n$, you are required to give the exact landing time of each airplane, in which the smallest gap between successive landings is maximized.

- md: the the minimum time gap
- x_i: the landing time for i^{th} airplane
- s_i: the starting point of time window for i^{th} airplane
- t_i: the ending point of time window for i^{th} airplane

Then, we can construct one LP problem:
For each i form the first one to the last one, where the biggest i+1 is the largest index,

$$\begin{aligned} max \quad & md \\ s.t. \quad & s_i \leq x_i \leq t_i \\ & x_{i+1} - x_i \geq md \\ & x_i, md \geq 0 \end{aligned}$$

If we adjust the problem to standard form:

$$\begin{aligned} min \quad & & -md \\ s.t. \quad & x_i & \leq t_i \\ & -x_i & \leq -s_i \\ & -x_{i+1} + x_i + md & \leq 0 \\ & x_i, md & \geq 0 \end{aligned}$$

We use x_i to denote the landing time of airplane i, and use z to denote the smallest gap. Then

official:

$$\begin{aligned} \max \quad & z & \\ s.t. \quad & x_i \geqslant s_i & \text{for all } i = 1, 2, \cdots, n \\ & x_i \leqslant t_i & \text{for all } i = 1, 2, \cdots, n \\ & x_{i+1} - x_i \geqslant z & \text{for all } i = 1, 2, \cdots, n-1 \end{aligned}$$

3 Interval Scheduling Problem

A teaching building has m classrooms in total, and n courses are trying to use them. Each course i $(i = 1, 2, \cdots, n)$ only uses one classroom during time interval $[S_i, F_i)$ $(F_i > S_i > 0)$. Considering any two courses can not be carried on in a same classroom at any time, you have to select as many courses as possible and arrange them without any time collision. For simplicity, suppose $2n$ elements in the set $\{S_1, F_1, \cdots, S_n, F_n\}$ are all different.

1. Please use ILP to solve this problem, then construct an instance and use GLPK or Gurobi or other similar tools to solve it.

2. If you relax the integral constraints and change ILP to an LP (e.g. change $x \in \{0,1\}$ to $0 \leqslant x \leqslant 1$), will solution of the LP contains only integers, regardless of values of all S_i and F_i? If it's true, prove it; if it's false, give a counter example. You can use the following lemma for help.

   LEMMA If matrix A has only 0, +1 or -1 entries, and each column of A has at most one +1 entry and at most one -1 entry. In addtion, the vector b has only integral entries. Then the vertex of polytope $\{x|Ax \leqslant b, x \geqslant 0\}$ contains only integral entries.

1. This question has at least five methods. In most methods, we can suppose $x_{ij} = \begin{cases} 1, & \text{course } i \text{ uses classroom } j \\ 0, & \text{otherwise} \end{cases} \quad (i, j = 1, 2, \cdots, n).$

- METHOD 3 Sort $S_1, F_1, \cdots, S_n, F_n$ ascendantly and get $T_1, \cdots, T_{2n}$, which will split the whole time window into $2n-1$ time slices. Suppose Ω_k denotes the set of courses that will occupy the time slice $[T_k, T_{k+1}]$. Use x_i to denote whether course i is chosen or not, then the LP is

$$\begin{array}{lrcll} \max & \sum_{i=1}^{n} x_i & & & \\ \text{s.t.} & \sum_{i \in \Omega_k} x_i & \leqslant & m & k = 1, \cdots, 2n-1 \\ & x_i & \in & \{0,1\} & i = 1, 2, \cdots, n \end{array}$$

P.S. 第二问 TA 也不会啊~~历史遗留问题*_*

4 Gas Station Placement

Let's consider a long, quiet country road with towns scattered very sparsely along it. Sinopec, largest oil refiner in China, wants to place gas stations along the road. Each gas station is assigned to a nearby town, and the distance between any two gas stations being as small as possible. Suppose there are n towns with distances from one endpoint of the road being $d_1, d_2, \cdots, d_n$. n gas stations are to be placed along the road, one station for one town. Besides, each station is at most r far away from its correspond town. $d_1, \cdots, d_n$ and r have been given and satisfied $d_1 < d_2 < \cdots < d_n$, $0 < r < d_1$ and $d_i + r < d_{i+1} - r$ for all i. The objective is to find the optimal placement such that the maximal distance between two successive gas stations is minimized.

Please formulate this problem as an LP.

- md: the maximal distance between two successive gas stations
- x_i: the distance for i^{th} gas station
- d_i: the correspond distance for i^{th} town
- r: the given constriction

Then, we can construct one LP problem:
For each i form the first one to the last one, where the biggest i+1 is the largest index,

$$\begin{aligned} min \quad & md \\ s.t. \quad & d_i - r \leq x_i \leq d_i + r \\ & d_{i+1} - d_i \leq md \\ & x_i, md \geq 0 \end{aligned}$$

If we adjust the problem to standard form:

$$\begin{aligned} min \quad & & md \\ s.t. \quad & x_i & \leq d_i + r \\ & -x_i & \leq -d_i + r \\ & d_{i+1} - d_i - md & \leq 0 \\ & x_i, md & \geq 0 \end{aligned}$$

We have formulated this issue as a LP problem.

5 Stable Matching Problem

n men $(m_1, m_2, \cdots, m_n)$ and n women $(w_1, w_2, \cdots, w_n)$, where each person has ranked all members of the opposite gender, have to make pairs. You need to give a stable matching of the men and women such that there is no unstable pair. Please choose one of the two following known conditions, formulate the problem as an ILP (*hint*: Problem 1.1 in this assignment), construct an instance and use GLPK or Gurobi or other similar tools to solve it.

1. You have known that for every two possible pairs (man m_i and woman w_j, man m_k and woman w_l), whether they are stable or not. If they are stable, then $S_{i,j,k,l} = 1$; if not, $S_{i,j,k,l} = 0$. $(i, j, k, l \in \{1, 2, \cdots, n\})$
2. You have known that for every man m_i, whether m_i likes woman w_j more than w_k. If he does, then $p_{i,j,k} = 1$; if not, $p_{i,j,k} = 0$. Similarly, if woman w_i likes man m_j more than m_k, then $q_{i,j,k} = 1$, else $q_{i,j,k} = 0$. $(i, j, k \in \{1, 2, \cdots, n\})$

Suppose $x_{ij} = \begin{cases} 1, & \text{man } i \text{ and woman } j \text{ get married} \\ 0, & \text{otherwise} \end{cases} \quad (i, j = 1, 2, \cdots, n).$

1. The ILP is as follows:

$$\begin{array}{lrcll} \min & 0 & & & \\ \text{s.t.} & \sum_{i=1}^{n} x_{ij} & = & 1 & \text{for all } j = 1, 2, \cdots, n \\ & \sum_{j=1}^{n} x_{ij} & = & 1 & \text{for all } i = 1, 2, \cdots, n \\ & x_{ij} + x_{kl} & \leqslant & S_{i,j,k,l} + 1 & \text{for all } i, j, k, l = 1, 2, \cdots, n, i \neq k, j \neq l \\ & x_{ij} & \in & \{0, 1\} & \text{for all } i, j = 1, 2, \cdots, n \end{array}$$

   The third constraint can be replaced by $x_{ij} + (1 - S_{i,j,k,l})x_{kl} \leqslant 1$.

2. If m_l likes w_k more than w_j, and w_k likes m_l more than m_i, then m_i and w_j will never become the wrecker if m_l and w_k get married (but we are not sure whether m_l and w_k will get married since other strong wreckers might exist). In other words, if $p_{l,k,j} = 1$ and $q_{k,l,i} = 1$, then $x_{ik} = 0$ and $x_{lj} = 0$. Based on this, the ILP is as follows:

$$\begin{array}{lrcll} \min & 0 & & & \\ \text{s.t.} & \sum_{i=1}^{n} x_{ij} & = & 1 & \text{for all } j = 1, 2, \cdots, n \\ & \sum_{j=1}^{n} x_{ij} & = & 1 & \text{for all } i = 1, 2, \cdots, n \\ & x_{ik} + x_{lj} & \leqslant & 3 - p_{l,k,j} - q_{k,l,i} & \text{for all } i, j, k, l = 1, 2, \cdots, n, k \neq j, l \neq i \\ & x_{ij} & \in & \{0, 1\} & \text{for all } i, j = 1, 2, \cdots, n \end{array}$$

   The third and fourth constraints can be replaced by $x_{ij} + x_{kl} \leqslant 2 - p_{ilj}q_{jki}$.

注：这里之所以约束 3 中为 3，是 p 和 q 同时满足时，<=1，可以有(1,0)(0,0)(0,1)出现，因为不排除某个 w 或 m 与四个人之外有极强的联系，这样与之建立关系之后，相互喜欢的 w 和 m 剩下的那个可以与单恋的异性建立稳定关系。

6 Duality

Please write the dual problem of the MULTICOMMODITYFLOW problem in *Lec8.pdf*, and give an explanation of the dual variables.

Please also construct an instance, and try to solve both primal and dual problem using GLPK or Gurobi or other similar tools.

For simplicity, we can assume that (u, v) denotes the arc $u \to v$. Then the primal can be rewritten and corrected as:

$$\begin{array}{lrcll} \max / \min & 0 & & & \\ \text{s.t.} & \sum_{i=1}^{k} f_i(u, v) & \leqslant & c(u, v) & \text{for each } (u, v) \\ & \sum_{v,(u,v)\in E} f_i(u, v) - \sum_{v,(v,u)\in E} f_i(v, u) & = & 0 & \text{for each } i \text{ and } u \in V \setminus \{s_i, t_i\} \\ & \sum_{v,(s_i,v)\in E} f_i(s_i, v) - \sum_{v,(v,s_i)\in E} f_i(v, s_i) & = & d_i & \text{for each } i \\ & f_i(u, v) & \geqslant & 0 & \text{for each } i, (u, v) \end{array}$$

If we use x_{uv} to denote the first constraints, y_{iu} the second and third constraints, then the duality is:

$$\begin{array}{lrcll} \min & c(u,v)x_{uv} + d_i y_{is_i} & & & \\ \text{s.t.} & x_{uv} + y_{iu} - y_{iv} & \geqslant & 0 & \text{for all } i \text{ and } u \neq t_i, v \neq t_i \\ & x_{ut_i} + y_{iu} & \geqslant & 0 & \text{for all } i, (u, t_i) \\ & x_{t_i v} - y_{iv} & \geqslant & 0 & \text{for all } i, (t_i, v) \\ & x_{uv} & \geqslant & 0 & \text{for all } (u,v) \end{array}$$

or

$$\begin{array}{lrcll} \max & c(u,v)x_{uv} + d_i y_{is_i} & & & \\ \text{s.t.} & x_{uv} + y_{iu} - y_{iv} & \leqslant & 0 & \text{for all } i \text{ and } u \neq t_i, v \neq t_i \\ & x_{ut_i} + y_{iu} & \leqslant & 0 & \text{for all } i, (u, t_i) \\ & x_{t_i v} - y_{iv} & \leqslant & 0 & \text{for all } i, (t_i, v) \\ & x_{uv} & \leqslant & 0 & \text{for all } (u,v) \end{array}$$

Assignment 5

1 Problem Reduction

Support the you are a matchmaker and there are some boys and girls. Since the boys are alway more than girls, you can assume that if a girl express her love to a boy , the boy will always accept her. Now you know every girl's thought(a girl may like more than one boy) and you want to make as much pairs as you can. show that you can do this using maximum flow algorithm.

Solution: Since every girl absolutely determined whether one pair is available, we firstly construct a network with direction: the nodes on left represent boys and nodes on right represent girls.Edges between them explain every girl's thought, which means if one girl love one boy, there will be an edge whose direction is from this boy to the girl.Now, we add one source which connects to every boy and one sink which is connected with every girl. The detailed graph is constructed as following:

- Edges:with direction from left to right, the flow value is confined to be integrated.
- Capacity:C(e)=1 for every edge for all $e \in E$, since every pair means one girl and one boy.
- Flow:with maximum flow algorithm, the maximal flow corresponds to a maximal matching.

(c)Complexity:

- Time:The flow value is at most equal to n_{boy},so time complexity is $O(mn_{boy})$.
- Space:It's easy to know space complexity is $O(size(network))$,which is $O(m+n)$

2 Problem Reduction

There is 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 satisfy the conditions.

2. (a)Algorithm:

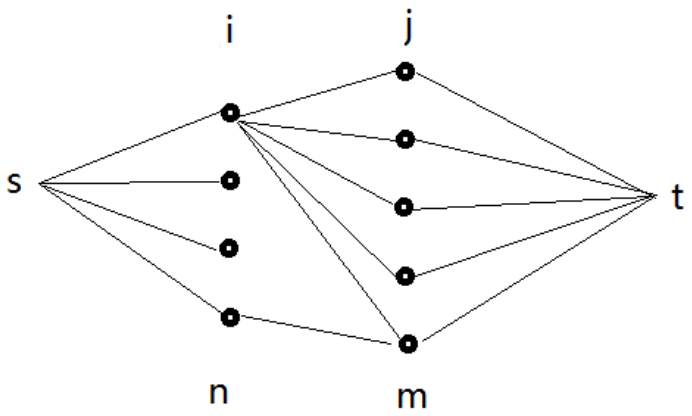
1. Construct network:assume matrix's size is $m * n$
   (a) Add super nodes s,t
   (b) Add edges:$s \to n_i$ for all $i = [1, ..., m]$;$n_i \to n_j$ for all $i = [1, ..., m]$,$j = [1, ..., n]$;$n_j \to t$ for all $j = [1, ..., m]$
   (c) Set $C(s \to n_i) = RowSum(i)$,$C(n_j \to t) = ColSum(j)$,$C(n_i \to n_j) = 1$

2. Solve the network as maximum flow problem
3. $f(n_i \to n_j) = Matrix[i][j]$ if maximum flow value is equal to $\sum RowSum(i)$, otherwise no feasible solution.

(b)Proof: It's easy to formulate the problem as Circulation problem,so we proved.

(c)Complexity:

- Time:The flow value is at most equal to mn,so time complexity is $O(m^2n^2)$.
- Space:It's easy to know space complexity is $O(size(network))$,which is $O(mn)$



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.

Solution: Apply Ford-Fulkerson Algorithm to given graph G, and get the maximal flow as well as the minimal cut (S, T).For every edge e = (u, v), $e \in G$, $u \in S, v \in T$, increase the C_e by unit value. Repeat the Ford-Fulkerson process then for every edge's step,compare the maximal flow result F_{plus} with original result$F_{original}$. If every F_{plus} is larger than $F_{original}$, we can claim that G has a unique minimum st cut.

```
UNIQUE CUT(G)
1  Flag = 0;
2  (F_original, E_cut) = Ford − Fulkerson(G);
3  for e = e ∈ E_cut
4       do
5            C_e = C_e + 1;
6            (F_plus, E) = Ford − Fulkerson(G);
7            if F_plus ≤ F_original
8              then
9                    Flag = 1; C_e = C_e − 1;
10 if (Flag == 1)
11   then
12           return False; Break;
13 return True;
```

Prove the Correctness: Suppose there is another minimal cut, thus there will be at least one edge the is not in E_{cut}.Because the chosen edge is adjusted with a larger C_e, if E_{cut} is unique, the flow through s-t cut will be increased.However, if another minimal cut is exist, at least one edge in E_{cut} is not used, which means when this edge is chosen, the result of Ford-Fulkerson algorithm will not change. Thus, it proves that the minimal cut is not unique.

Analyse the Complexity: Ford-Fulkerson algorithm is with complexity of O(mC),the number of edges in $E_{cut}$is a constant K,$T(n) = O(K(mC + c1) + c2) = O(mC)$.

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 NO be walk through more than once.

4. (a)Algorithm:

1. Construct network:
   (a) Node:each matrix item is a node.s is corresponding to $M[1][1]$,t is corresponding to $M[m][n]$.For each node $n_{i,j}$ except s,t,add a node $c_{i,j}$
   (b) Add edges:$s \to n_{1,2}$,$s \to n_{2,1}$,$n_{m-1,n} \to t$,$n_{m,n-1} \to t$;$n_{i,j} \to c_{i,j}$;$c_{i,j} \to n_{i,j+1}, c_{i,j} \to n_{i+1,j}$
   (c) Set $cost(u \to n_{i,j}) = M[i][j]$,$cost(u \to c_{i,j}) = 0$;set C(e) = 1;
2. Solve the network as minimum cost problem with $v = 2$.

(b)Proof:

- A node is at most visited once since there is only one edge out of a node with $C = 1$.
- There is 2 disjoint path from s to t,since $v = 2$ and $C = 1$,one is for walk right and down,the inverse of the other is for up and left.
- The cost is minimum.

(c)Complexity:

- Time:The same as minimum cost problem,$O(mnC)$.
- Space:It's easy to know space complexity is $O(size(network))$,which is $O(mn)$

INPUT:
a network $G =< V, E >$, where each edge e has a capacity $C(e) > 0$, and a cost $w(e)$ for transferring a unit through edge e. Two special node: source s and sink t. A flow value v_0.
OUTPUT:
to find a circulation f with flow value v_0 and the cost is minimized.

5 Dogs and kennels

On a grid map there are n little dogs and n kennels. In each unit time, every little dog can move one unit step, either horizontally, or vertically, to an adjacent point. For each little dog, you need to pay a 1 travel fee for every step it moves, until it enters a kennel. The task is complicated with the restriction that each kennel can accommodate only one little dog.

Give a polynomial-time algorithm to compute the minimum amount of money you need to pay in order to send these n little dogs into those n different kennels.

Solution: In this issue, we firstly transform the map to effective information about how many steps for every dog reach every kennel.At the same time, we record the maximal steps and minimal steps for every dog gets into one kernel and every kernel gets one dog being into it.Then, we construct one two layers network, the first layer has nodes of the number of dogs, and the second layer has nodes of the number of kernels, which corresponding to every dog and kernel. We add one source point s and one sink point t, then connect s to every dog layer points, with capacities of the possible steps from specific dog to one kernel [min-step-for-this-dog, max-step-for-this-dog].t is connected with every kernel layer node, whose capacities are [min-step-for-this-kernel, max-step-for-this-kernel] for every node. Every node in dog layer is connected to the possible kernel it may reach, the capacity constrain is the step number form this dog to this kernel

- Edges:with direction from left to right, the flow value is confined to be integrated, because it represent the number of steps.
- Capacity:the edges connected to s and t is both have lower bound and upper bound.
- Flow:with maximum flow algorithm, the maximal flow corresponds to one required assignment for dogs and kernels.However, since we want to find the minimal steps number, we should set the capacities to be negative number.

其实还有一个更简单的办法：第一层全部的狗，第二层全部的狗舍，前后加上 s 和 t，s 到狗 c=1，狗到狗舍 c=1，cost=到对应狗舍的距离，狗舍到 t，c=1。求最小 cost 流。

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.

Solution:

First we give the solution to the problem. Then we give intuition about how one might discover the network that solves the problem. Given a graph $G = (V, E)$ and a density α, we wish to discover whether there is a subset $W \subseteq V$ such that $e(W)/|W| \geq \alpha$, where $e(W)$ is the number of edges with both endpoints in W. We create a network flow to solve the problem as follows. Add a source node s and connect it to each vertex by an edge of weight $|E|$, and add a sink node t and connect it to each vertex v_i by an edge of weight $|E|-d_i + 2\alpha$, where d_i is the degree of vertex v_i. Note that all edges have positive weight, since $d_i \leq |E|$.

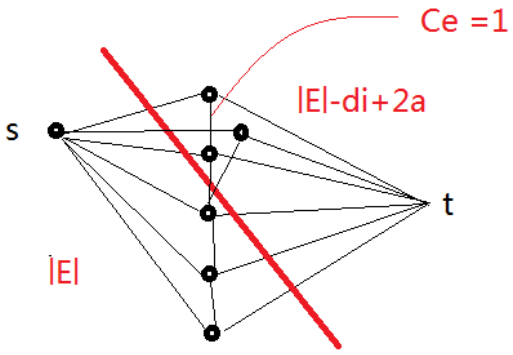
Find the minimum weight cut in this graph, and let V_1 be the set of vertices connected to s and V_2 be the set of vertices connected to t in this cut. Then if $V_1 = \emptyset$, there is no subset of V of density greater than α. Otherwise, V_1 has density greater than α.

We prove this as follows. We know that any cut in the graph must have weight greater than or equal to the cut that separates V_1 from V_2. Hence in particular the cut isolating s has weight greater than the weight of the minimum cut. We let $c_{i,j} = 1$ if edge (i, j) is in G.

因为是最小割，所以一定有边的权重更大，C=1 化权重为边数

Thus

$$\begin{aligned} |V||E| &\geq \sum_{i\in V_2}|E| + \sum_{j\in V_1}(|E| - d_j + 2\alpha) + \sum_{i\in V_2, j\in V_1} c_{i,j} \\ &\geq |V||E| + 2\alpha|V_1| - \sum_{j\in V_1}(d_j - \sum_{i\in V_2} c_{i,j}) \\ 0 &\geq 2\alpha|V_1| - 2e(V_1) \\ \frac{e(v_1)}{|V_1|} &\geq \alpha. \end{aligned}$$



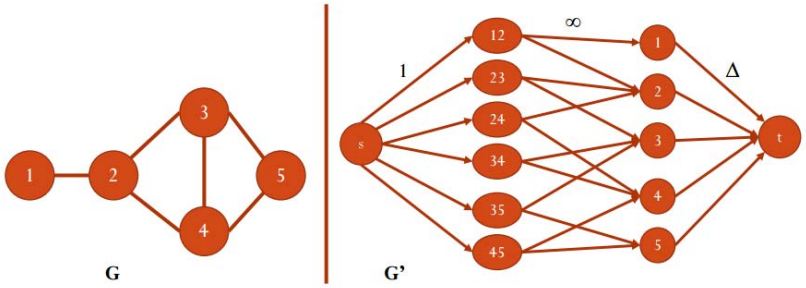
On the other hand, if $V_1 = \emptyset$, we know that the inequality above is reversed, so we know $e(V_1)/|V1| \leq \alpha$ for any set $V_1 \subseteq V$.

Why should we have expected this network to solve the problem? The symmetries of the problem imply that all nodes should be connected to the source and to the sink, and that the only pieces of information that we can use to distinguish the weights of these links must be the degrees of the nodes. We wish the procedure to isolate vertices that are highly connected on one side, and ill connected on the other side. Hence by making the links to the sink depend on the difference between 2α and the degree, we encourage nodes of high degree to connect to the source, and nodes of low degree to connect to the sink. We add a constant to the edges to the source and the edges to the sink to ensure all edges have positive weight. And the connectivity of vertices inside the graph ensures that nodes of moderate degree connected to high degree nodes also segregate into the part of the partition with their high degree neighbors.

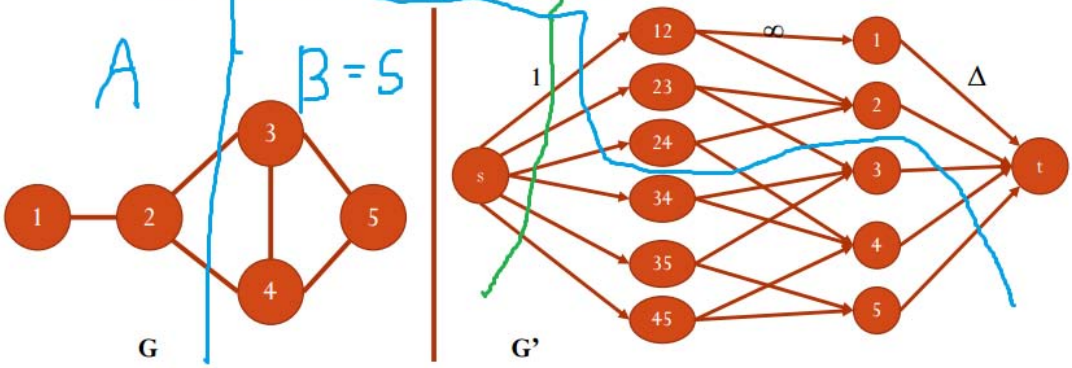
补充方法 2：另一种理解的思路，只是部分摘选，详情可自己 Google

Network Flow: Applications

- We construct the following network graph **G'**.
  - There is a vertex corresponding to every vertex in **G**.
  - There is a vertex $\mathbf{v_{ij}}$ corresponding to each *edge* in **G**.
  - There is a source **s** and a sink vertex **t**.
  - Vertex $\mathbf{v_{ij}}$ has edges to vertex **i** and **j**. These have capacity ∞.
  - There is an edge from **s** to all vertices $\mathbf{v_{ij}}$. These have capacity **1**.
  - There is an edge from every vertex **v** to **t** with capacity Δ.



- Let **(A,B)** be a min-cut in **G'**. Let **S** be the vertices on the right that are in **B**.
- Claim 1: If $\mathbf{v_{ij}}$ is in **A**, then both **i** and **j** are in **A**.
- Claim 2: If **i** and **j** is in **A**, then $\mathbf{v_{ij}}$ is in **A**.
- Claim 3: $\mathbf{c(A,B) = |E| - e(S) + |S|*\Delta}$
- Claim 4: There is a subset **S** with cohesiveness > Δ if and only if the min-cut in **G'** has capacity < **|E|**.



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 fP be the amount of flow on path P. Then maximum flow says to find

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

(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.

minimize

$$\Sigma u_e y_e$$

subject to

$$\Sigma_{e \in P} y_e \geq 1, \text{for all path } P$$
$$y \geq 0$$

If all the y's are 0 or 1, each y corresponds to an edge, the objective function calculates the minimum cut (S, T), $y = 1$ means that this edge has one end in S and the other in T. Each constraint corresponds to a path. The left side of a constraint lists all edges on the path. Because the path is from s to t, at least one edge on the path must be cut, so the sum is at least 1. Since the LP is the dual to a maximum flow problem, an integral solution must exist.

另一种把最大流问题形式化成线性规划问题是通过流分解。考虑网络中所有的路径 s-t, 用 fP 来表示路径 P 上的流。那么最大流就是求：

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

写出上述线性规划问题的对偶形式，并用英语解释目标函数和约束。

1 对偶形式

目标函数：

min $\sum u_e y_e$

约束：

$\sum_{e \text{ in } P} y_e \geq 1$ for all path P

$y_e \geq 0$

3 解释

a) 目标函数，y_e 表示一条边，0 表示不选，1 表示选，表示最小割

b) 约束表示每一条从 s 到 t 得路径中，至少有一条边在割上

Assignment 6

1 Integer Programming

Given an integer $m \times n$ matrix A and an integer m-vector b, the Integer programming problem asks whether there is an integer n-vector x such that $Ax \geq b$. Prove that Integer-programming is in NP-complete.

Solution:we can prove that Integer-programing is NPC in two steps:

- We can prove that Integer-programing is NP
- As we known, 3SAT is NPC, we can finish our proof through the proving that 3SAT$\leq_p$ Integer-programing

Part1. Certificate:the n-vector x with integer elements.Thus, we can verify the constrain of Ax $\geq$ b in polynomial time.
Part2. Now our goal is to show that 3SAT is polynomially reducible to the Integer-programing, which means that for one input of instance in 3SAT, the corresponding output is satisfied if and only if there is an integer x vector that can satisfy the requirement of Ax $\geq$ b.If the clauses are satisfiable, we regard it as one truth assignment.Then, we can treat every row of A multiply x to one element of b as one constrain, and the factors of this row can be constructed as a clause of 3SAT.

For example, there are 4 varibles in vector of x, x_1, x_2, x_3, x_4, the corresponding varibles of y is y_i.If there is one clause is $y_1 \vee y_3 \vee \neg y_4$, the corresponding constrain of x is $x_1 + x_3 + (1 - x4) \geq 1$.We set x_i to be 1 or 0 corresponding to true or false value of y_i.We can construct the matrix of A row by row according to each clause. Then we can move the constant element to the right to construct one element of vector b.Clearly, if one assignment is true, both the clause and the constrain of matrix will be satisfied.Thus, 3SAT is polynomially reducible to the Integer-programing.

2 Mine-sweeper

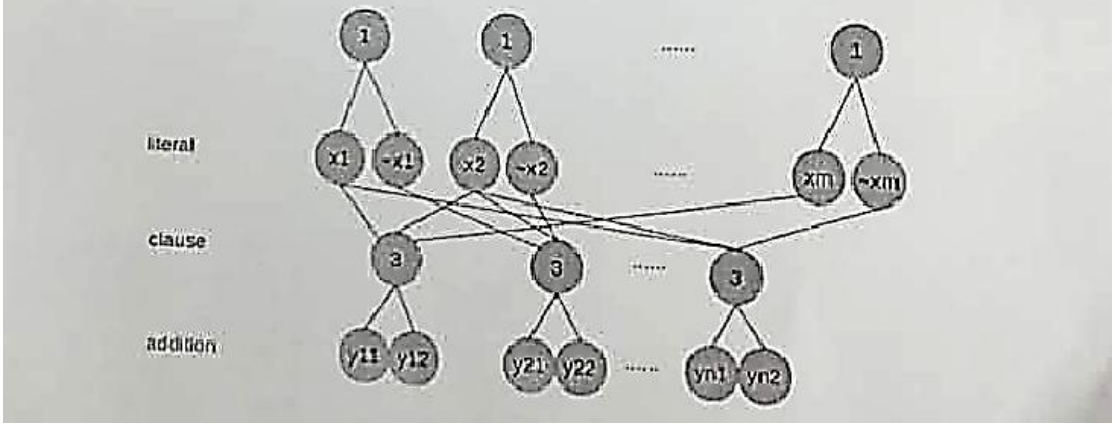
This problem is inspired by the single-player game Mine-sweeper, generalized to an arbitrary graph. Let G be an undirected graph, where each node either contains a single, hidden mine or is empty. The player chooses nodes, one by one. If the player chooses a node containing a mine, the player loses. If the player chooses an empty node, the player learns the number of neighboring nodes containing mines. (A neighboring node is one connected to the chosen node by an edge.). The player wins if and when all empty nodes have been so chosen.

In the **mine consistency problem**, you are given a graph G, along with numbers labeling some of G's nodes. You must determine whether a placement of mines on the remaining nodes is possible, so that any node v that is labeled m has exactly m neighboring nodes containing mines. Formulate this problem as a language and show that it is NP-complete.

Question 2

Proof: Obviously *Mine-Sweeper* $\in NP$, because a given placement of mines can be determined possible or not.

Prove 3-*SAT* $\leq_p$ *Mine-Sweeper*.
A 3-*SAT* problem: literal $\{x_1, x_2 \cdots x_m\}$,CNF $C_1 \wedge C_2 \wedge \cdots \wedge C_n$. Formulate a *Mine-Sweeper* problem. The graph is shown below.



注：最下边的 y_n1, y_n2 可以理解为填充完 xi 和–xi 之后添补的变量，保证 3 的存在有意义。

3 Half-3SAT

In the Half-3SAT problem, we are given a 3SAT formula ϕ with n variables and m clauses, where m is even. We wish to determine whether there exists an assignment to the variables of ϕ such that exactly half the clauses evaluate to false and exactly half the clauses evaluate to true. Prove that Half-3SA problem is in NP-complete.

Solution:we can prove that Half-3SAT problem is NPC in two steps:

- We can prove that Half-3SAT problem is NP
- As we known, 3SAT is NPC, we can finish our proof through the proving that 3SAT$\leq_p$ Half-3SAT problem

Part1. Certificate:for one regular Half-3SAT problem,if there are n variables and m clauses, we can set their value to be true or false,according to the formulation of clauses, we can get its boolean value.If half of the values are true, then it will be one true assignment.Clearly, we only need to set the value for every variable once, this process will be finished in polynomial time.

Part2. Now our goal is to show that 3SAT is polynomially reducible to Half-3SAT problem, which means that for one input of instance in 3SAT, with one transformation, there will be one instance of Half-3SAT problem. What's more, the instance of 3SAT will be satisfied if and only if the instance of Half-3SAT problem is satisfied.

For one 3SAT problem with n variables and m clauses, the value of $C_1 \wedge C_2 \wedge C_3 \wedge ...C_m$ is defined to be ϕ.The corresponding Half-3SAT problem is 4m clauses.The first part of these clauses are the same as ϕ, then there will be m clauses which formulation is $s_1 \vee s2 \vee \neg s2$.Clearly these clauses are always true no matter what kind of assignment it is.At last, the left 2m clauses are constructed to be all false or all true.For example, $s_1 \vee s2 \vee s3$ is the formulation of all of clauses.

Now we can prove this construction satisifies the second requirement:

- if 3SAT is satisfied, ϕ is true, half of the clauses in Half-3SAT is true, then, the last 2m clauses are set to be false,the requirement of Half-3SAT is satisfied, the true assignment of ϕ is transformed to a ideal Half-3SAT assignment.
- if Half-3SAT is satisfied, because the last 2m clauses must to be false, or 3/4 of clauses are true and the requirement can not be satisfied.Thus, the first m clauses are true.We can take the assignment of first m clauses as 3SAT assignment.Clearly, the corresponding requirement is satisfied.

4 Solitaire Game

In the following solitaire game, you are given an $n \times n$ board. On each of its n^2 positions lies either a blue stone, a red stone, or nothing at all. You play by removing stones from the board so that each column contains only stones of a single color and each row contains at least one stone. You win if you achieve this objective.

Winning may or may not be possible, depending upon the initial configuration. You must determine the given initial configuration is a winnable game configuration. Let **SOLITAIRE** $= \{< G > | G$ is a winnable game configuration$\}$. Prove that **SOLITAIRE** is NP-complete.

Proof: Obviously *Solitaire-Game* is NP.

Prove $3SAT \leqslant_p$ *Solitaire-Game*.
For any $3SAT$ problem, containing m variables and n clauses, if $m < n$, create $n - m$ variables, which will not used; else if $m > n$, create $m - n$ clauses like $(x_1 \vee \neg x_1 \vee x_2)$, which will not affect the result of $3SAT$ problem.
Assume $n' = max\{m, n\}$, the new $3SAT$ problem has n' variables and n' clauses. Create a $n' \times n'$ board. Every row represents a variable, and every column represents a clause. If the j-th clause have literal x_i, put red stone in (i, j); if the j-th clause have literal $\neg x_i$, put blue stone in (i, j). Thus we generate a *Solitaire-Game* based on $3SAT$ problem. If this *Solitaire-Game* have a solution, observe the i-th row, if all stones are red, $x_i = true$; else if all stones are blue, $x_i = false$; else if no stone are at this row, either *true* or *false* will be OK. End of proof.

1， 冗余变量、冗余子句补全成方针
2， 每一行一个变量，对、错或者不用
3， 每一列一个子句，至少用一个变量

5 Directed Disjoint Paths Problem

The **Directed Disjoint Paths Problem** is defined as follows. We are given a directed graph G and k pairs of nodes $(s_1, t_1), (s_2, t_2), \cdots, (s_k, t_k)$. The problem is to decide whether there exist node-disjoint paths $P_1, P_2, \cdots, P_k$

so that P_i goes from s_i to t_i. In details, the node-disjoint paths means that P_i and P_j $(1 \leq i \leq k, 1 \leq j \leq k, i \neq j)$share no nodes.

Show that **Directed Disjoint Paths** is NP-complete.

DISJOINT CONNECTING PATHS:

Given a graph G and k pairs of vertices (s_i, t_i), are there k vertex disjoint paths $P_1, P_2, \ldots, P_k$ such that P_i connects s_i with t_i ?

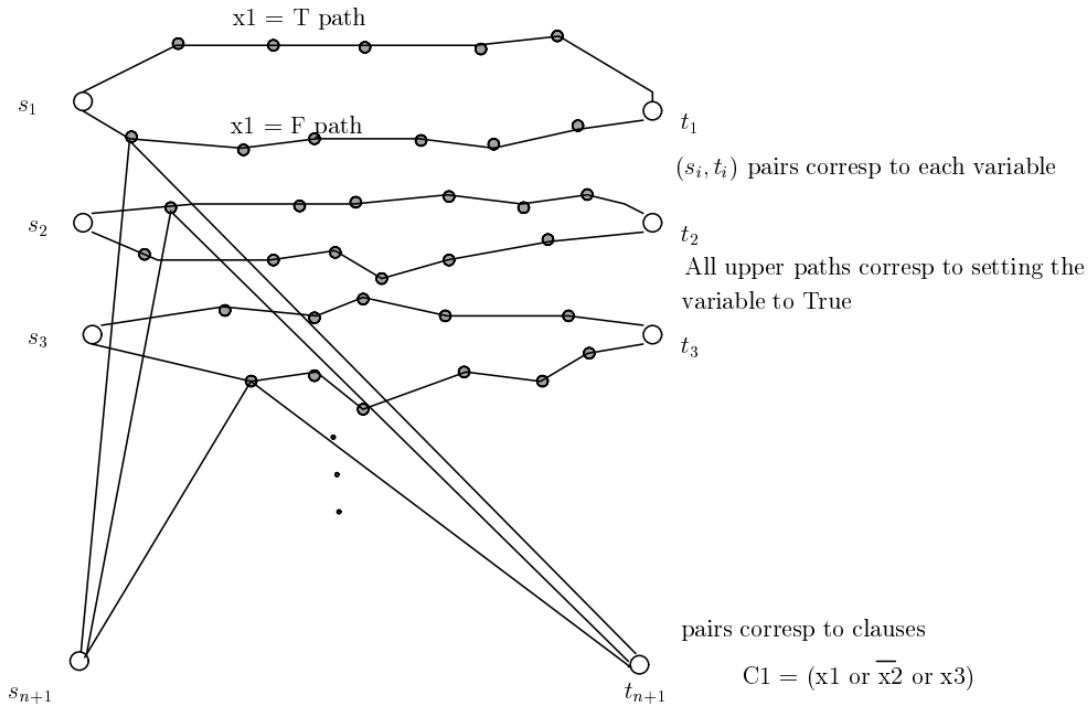
This problem is NP-complete as can be seen by a reduction from 3SAT.

Corresponding to each variable x_i, we have a pair of vertices s_i, t_i with two internally vertex disjoint paths of length m connecting them (where m is the number of clauses). One path is called the *true* path

and the other is the *false* path. We can go from s_i to t_i on either path, and that corresponds to setting x_i to true or false respectively.

For clause $C_j = (x_i \wedge \overline{x_\ell} \wedge x_k)$ we have a pair of vertices s_{n+j}, t_{n+j}. This pair of vertices is connected as follows: we add an edge from s_{n+j} to a node on the false path of x_i, and an edge from this node to t_{n+j}. Since if x_i is true, the s_i, t_i path will use the true path, leaving the false path free that will let s_{n+j} reach t_{n+j}. We add an edge from s_{n+j} to a node on the true path of x_ℓ, and an edge from this node to t_{n+j}. Since if x_ℓ is false, the s_ℓ, t_ℓ path will use the false path, leaving the true path free that will let s_{n+j} reach t_{n+j}. If x_i is true, and x_ℓ is false then we can connect s_{n+j} to t_{n+j} through either node. If clause C_j and clause $C_{j'}$ both use x_i then care is taken to connect the s_{n+j} vertex to a distinct node from the vertex $s_{n+j'}$ is connected to, on the false path of x_i. So if x_i is indeed true then the true path from s_i to t_i is used, and that enables both the clauses C_j and $C_{j'}$ to be true, also enabling both s_{n+j} and $s_{n+j'}$ to be connected to their respective partners.

Proofs of this construction are left to the reader.



注：s1 – sn 每个代表一个变量，sn+1 – sn+m 每个代表一个子句。

6 Longest Common Subsequence Problem

Given a finite sequence $S = s_1, s_2, \cdots, s_m$, we define a subsequence S' of S to be any sequence which consists of S with between 0 and m terms deleted (e.g. *ac*, *ad*, and *abcd* are all subsequences of *abcd*). Given a set $R = \{S_1, S_2, \cdots, S_p\}$ of sequences, we speak of a *Longest Common Subsequence* of R, LCS(R), as a longest sequence S such that is S is a subsequence of S_i, for $i = 1, \cdots, p$. For example, abe = LCS({ababe, cabe, abdde}).

The *yes/no* **Longest Common Subsequence Problem (LCS)** is: Given an integer k and a listing of the sequences $R = \{S_1, S_2, \cdots, S_p\}$, is $|\text{LCS}(R)| \geqslant k$, where $|S|$ is the number of terms in sequence S?

Show that *yes/no* **LCS** problem is NP-complete.

Proof. We prove the hardness of the problem by a reduction from 3-SAT.

Given a 3-SAT instance with variables $x_1, x_2, \ldots, x_k$ and clauses $c_1, c_2, \ldots, c_l$, we construct an instance of C-LCS with two input strings and $k+l-1$ constraints. The alphabet of A_1 and A_2 is the set of clauses $c_1, c_2, \ldots, c_l$ and a set of separators $\{s_1, s_2, \ldots, s_{k-1}\}$ separating between the variables.
We construct A_1 as follows. For each variable x_i we create a substring X_i by setting all the clauses satisfied with $x_i = true$ followed by all the clauses satisfied with $x_1 = false$ (we set the clauses in a sorted order). We then set A_1 to be $X_1 S_1 X_2 S_2 \ldots S_{k-1} X_k$, the X_i substrings separated by the appropriate separators.
We similarly construct A_2. We create a substring X_i' by setting all the clauses satisfied with $x_i = false$ followed by all the clauses satisfied with $x_1 = true$ (we set the clauses in a sorted order). We then set A_2 to be $X_1' S_1 X_2' S_2 \ldots S_{k-1} X_k'$, the X_i' substrings separated by the appropriate separators.
Let $c_1, c_2, \ldots, c_l$ and $s_1, s_2, \ldots, s_{k-1}$ be the group of constraints. Note that, all of them are of length one.

1， 两个序列，一个由依次 x_i 设定为 1 所导致的正确子句在前，0 在后+间断排列而成，另一个由 x_i 设定为 0 所导致的正确句子在前，1 在后+间断排列而成。

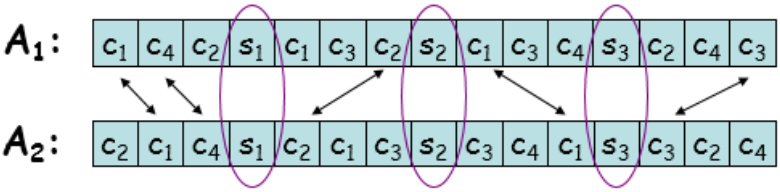
$$(X_1 \vee X_2 \vee X_3) \wedge (\bar{X}_1 \vee \bar{X}_2 \vee X_4) \wedge (X_2 \vee \bar{X}_3 \vee \bar{X}_4) \wedge (X_1 \vee \bar{X}_3 \vee X_4)$$

A_1:	c_1	c_4	c_2	s_1	c_1	c_3	c_2	s_2	c_1	c_3	c_4	s_3	c_2	c_4	c_3

A_2:	c_2	c_1	c_4	s_1	c_2	c_1	c_3	s_2	c_3	c_4	c_1	s_3	c_3	c_2	c_4

Feasible 3 – SAT instance

$$(X_1 \vee X_2 \vee X_3) \wedge (\bar{X}_1 \vee \bar{X}_2 \vee X_4) \wedge (X_2 \vee \bar{X}_3 \vee \bar{X}_4) \wedge (X_1 \vee \bar{X}_3 \vee X_4)$$



一个正确的指派会分别从 A1 和 A2 中分别选择对应向量，满足总子句数量的那个正是最长的子集链。

注：这个摘录存在不全的问题，不太容易理解，Google 上有 LCS 问题 3SAT 处理的论文，可以具体参考。

Assignment 7

1 Bin Packing

Bin Packing is as follows: Given n items with sizes $a_1, \cdots, a_n \in (0, 1]$, find a packing in unit-sized bins that minimizes the number of bins used.

Give a 2-approximation algorithm for this problem and analysis the approximation factor.

解:(a) 算法如下:

新建一个Bin b 和一个大顶堆 H, Bin 的值定义为其剩余容量 c. 将 b 插入 H 中。
for i=0 to n **do**
  $t = H$ 的堆顶元素。
  if a_i 能够放入 H 的堆顶的Bin b 中 **then**
    将 a_i 放入 b 中;
    $c(b) = c(b) - a_i$;
  else
    新建一个Bin t 将 a_i 放入 t 中
    $c(t) = 1 - a_i$;
    将 t 插入 H 中;
  end if
end for

首先不会有两个罐的剩余容量同时少于1/2,因为否则,第二个罐的物品会直接放入每一个罐中。根据算法,不会新建一个罐。

- 若每个罐的容量都大于或等于1/2,
  则 $\sum_{i=1}^{k} \geq k/2$. 故 $k \leq 2\sum_{i=1}^{n} a_i \leq 2\lceil \sum_{i=1}^{n} a_i \rceil \leq 2B^*$
- 若最后一罐的容量小于1/2,则
  $\sum_{i=1}^{k} a_i = \sum_{i=1}^{k-1} v_i + v_k > \sum_{i=1}^{k-1}(1 - v_k) + v_k > (k-1) - (k-2)v_k > k/2.$

注：另一种解释，考虑每两个相邻的箱子，相加的和一定大于两个 1/2 的和，否则可以合并，则有 k/2 * (1/2+1/2) < $(\sum_{i=1}^{k} Ci)/1$ = OPT 所以有 k < 2OPT

2 Steiner Tree Problem

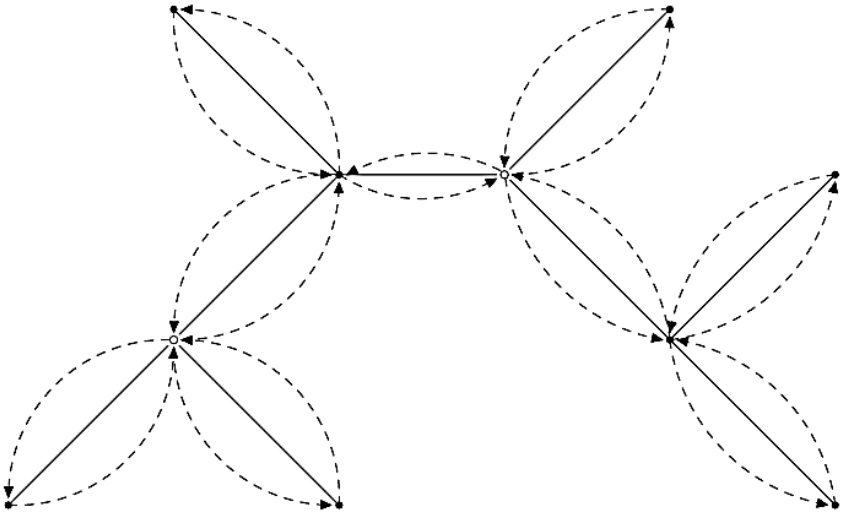
Given an undirected graph $G = (V, E)$ with edge costs and set $T \subseteq V$ of required vertices, the *Steiner Tree Problem* is to find a minimum cost tree in G containing every vertex in T (vertices in $V - T$ may or may not be used in T).

Give a 2-approximation algorithm if G is complete and the edge costs satisfy the triangle inequality.

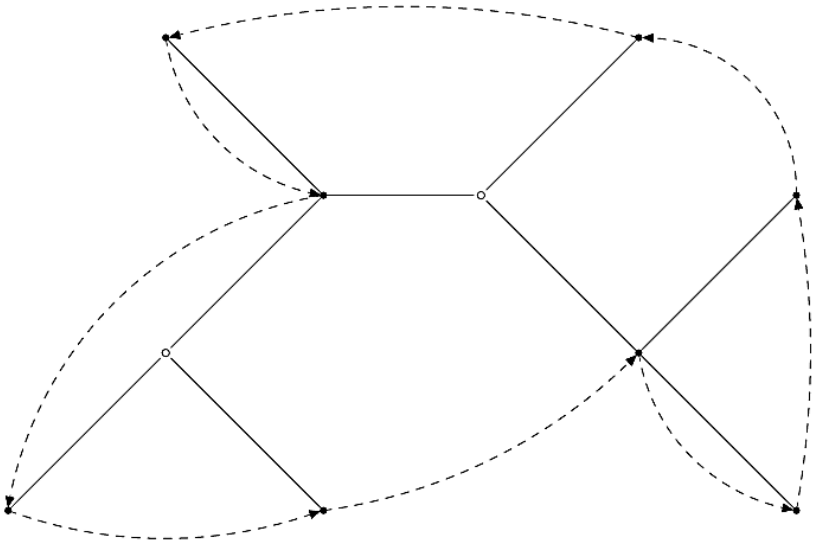
Solution: (a) We call the restricted (by the triangle inequality) problem *metric Steiner tree problem*

Let R denote the set of required vertices. Clearly, a minimum spanning tree (MST) on R is a feasible solution for this problem. We will prove that the cost of an MST on R is within $2 \cdot \text{OPT}$.

Consider a Steiner tree of cost OPT. By doubling its edges we obtain an Eulerian graph connecting all vertices of R and, possibly, some Steiner vertices. Find an Euler tour of this graph, for example by traversing the vertices in DFS (depth first search) order:



The cost of this Euler tour is $2 \cdot \text{OPT}$. Next obtain a Hmiltom cycle on the vertices of R by traversing the Euler tour and "short-cutting" Steiner vertices and previously visited vertices of R:



Because of triangle inequality, the shortcuts do not increase the cost of the tour. If we delete one edge of this Hamiltonian cycle, we obtain a path that spans R and has cost at most $2 \cdot \text{OPT}$. This path is else a spanning tree on R. Hence, the MST on R has cost at most $2 \cdot \text{OPT}$.

注：核心思想是对于 T 中点形成的完全树中任意两个点，若用 T 之外的 G 之中的点用两条边分别相连首尾，有两边之和大于第三边（两倍情况下可对所有树中的边进行不等式讨论）

3 Vertex Cover

Consider the following algorithm for (unweighted) **Vertex Cover**: In each connected component of the input graph execute a depth first search (DFS). Output the nodes that are not the leaves of the DFS tree.

Show that the output is indeed a vertex cover, and that it approximates the minimum vertex cover within a factor of 2.

解：只需考虑连通分支数为1的情况。若图G有多个连通分支，那么每个连通分支的顶覆盖合起来就是G的顶覆盖。假设G的第i个连通分支的顶覆盖为C_i，且$|C_i|/|C_i^*| \leq 2$，其中C_i^*是第i个连通分支的最小顶覆盖，那么$\sum_{i=1}^{n}|C_i| \leq 2\sum_{i=1}^{n}|C_i^*|$. 故$|\cup_{i=1}^{n} C_i|/|C^*| \leq 2$.

下面考虑G连通的情况。首先证明该算法输出是一个顶覆盖。$\forall e = (u,v) \in E(G)$，在DFS树中，$u,v$不能同时是叶子，从而要么$u \in A$或者$v \in A$或者两者都是$A$中的元素，这里$A$表示算法输出的顶点集合。故$e$被$A$覆盖。

然后证明$|A| \leq 2|C^*|$. 我们通过与另一个顶点覆盖算法建立关系来证明。该算法来自《算法导论》（第二版，中译本，第 *634* 页）

APPROX-VERTEX-COVER(G)

1: $C \leftarrow \emptyset$;
2: $E' \leftarrow E(G)$;
3: **while** $E' \neq \emptyset$ **do**
4: let (u,v) be an arbitrary edge of E';
5: $C \leftarrow C \cup \{u,v\}$;
6: remove from E' every edge incident on u or v;
7: **end while**
8: **return** C;

设该算法的输出为C. 课本中已证$|C|/|C^*| \leq 2$. 算法第4行如果按照DFS的搜索顺序给出边$(u,v) \in E'$，那么有$A \subseteq C$，故$|A| \leq |C|$. 从而$|A|/|C^*| \leq |C|/|C^*| \leq 2$.

4 MAX-3SAT

Given a set of clauses $C_1, \cdots, C_n$, each of length 3, and you need to find an assignment to maximize the number of satisfied clauses.

Please find a approximation algorithm to solve the problem and give the approximation factor.

7.1 Max Exact 3SAT

We remind the reader that an instance of 3SAT is a boolean formula, for example $F = (x_1 + x_2 + x_3)(x_4 + \overline{x_1} + x_2)$, and the decision problem is to decide if the formula has a satisfiable assignment. Interestingly, we can turn this into an optimization problem.

Problem: Max 3SAT

Instance: A collection of clauses: $C_1, \ldots, C_m$.
Output: Find the assignment to $x_1, ..., x_n$ that satisfies the maximum number of clauses.

Clearly, since 3SAT is NP-Complete it implies that Max 3SAT is NP-Hard. In particular, the formula F becomes the following set of two clauses:

$$x_1 + x_2 + x_3 \quad \text{and} \quad x_4 + \overline{x_1} + x_2.$$

Note, that Max 3SAT is a *maximization problem*.

Definition 7.1.1 Algorithm **Alg** for a maximization problem achieves an approximation factor α if for all inputs, we have:

$$\frac{\mathbf{Alg}(G)}{\mathrm{Opt}(G)} \geq \alpha.$$

In the following, we present a ***randomized algorithm*** – it is allowed to consult with a source of random numbers in making decisions. A key property we need about random variables, is the linearity of expectation property, which is easy to derive directly from the definition of expectation.

Definition 7.1.2 (*Linearity of expectations*.) Given two random variables X, Y (not necessarily independent, we have that $\mathbf{E}[X + Y] = \mathbf{E}[X] + \mathbf{E}[Y]$.

Theorem 7.1.3 *One can achieve (in expectation)* $(7/8)$*-approximation to Max 3SAT in polynomial time. Namely, if the instance has m clauses, then the generated assignment satisfies* $(7/8)m$ *clauses in* expectation.

Proof: Let $x_1, \ldots, x_n$ be the n variables used in the given instance. The algorithm works by randomly assigning values to $x_1, \ldots, x_n$, independently, and equal probability, to 0 or 1, for each one of the variables.

Let Y_i be the indicator variables which is 1 if (and only if) the ith clause is satisfied by the generated random assignment and 0 otherwise, for $i = 1, \ldots, m$. Formally, we have

$$Y_i = \begin{cases} 1 & C_i \text{ is satisfied by the generated assignment,} \\ 0 & \text{otherwise.} \end{cases}$$

Now, the number of clauses satisfied by the given assignment is $Y = \sum_{i=1}^{m} Y_i$. We claim that $\mathbf{E}[Y] = (7/8)m$, where m is the number of clauses in the input. Indeed, we have

$$\mathbf{E}[Y] = \mathbf{E}\left[\sum_{i=1}^{m} Y_i\right] = \sum_{i=1}^{m} \mathbf{E}[Y_i]$$

by linearity of expectation. Now, what is the probability that $Y_i = 0$? This is the probability that all three literals appear in the clause C_i are evaluated to FALSE. Since the three literals are instance of three distinct variable, these three events are independent, and as such the probability for this happening is

$$\mathbf{Pr}[Y_i = 0] = \frac{1}{2} * \frac{1}{2} * \frac{1}{2} = \frac{1}{8}.$$

(Another way to see this, is to observe that since C_i has exactly three literals, there is only one possible assignment to the three variables appearing in it, such that the clause evaluates to FALSE. Now, there are eight (8) possible assignments to this clause, and thus the probability of picking a FALSE assignment is $1/8$.) Thus,

$$\mathbf{Pr}[Y_i = 1] = 1 - \mathbf{Pr}[Y_i = 0] = \frac{7}{8},$$

and

$$\mathbf{E}[Y_i] = \mathbf{Pr}[Y_i = 0] * 0 + \mathbf{Pr}[Y_i = 1] * 1 = \frac{7}{8}.$$

Namely, $\mathbf{E}[\#\text{ of clauses sat}] = \mathbf{E}[Y] = \sum_{i=1}^{m} \mathbf{E}[Y_i] = (7/8)m$. Since the optimal solution satisfies at most m clauses, the claim follows. ■

Curiously, Theorem 7.1.3 is stronger than what one usually would be able to get for an approximation algorithm. Here, the approximation quality is independent of how well the optimal solution does (the optimal can satisfy at most m clauses, as such we get a $(7/8)$-approximation. Curiouser and curiouser[2], the algorithm does not even look on the input when generating the random assignment.

----之后利用确定算法可以求出具体的 7/8 证明。

----此题可以有一个简单的两倍近似算法：每次随机选择一个变量，如 xi，将得到正确的子句和错误的子句选出来，选择其中更多的一部分（该部分对应的那个变量的 0/1 取值）。对于剩下的子句，再选择另一个变量，重复该操作，直到全部的子句都被这样选择完。我们得到了一组赋值和一组对应的子句，这个子句集的大小就是我们输出的最大满足数。

证明 2 倍：对于 OPT 有一个明显的上界，就是全部的子句数量。而我们的近似算法有一个下界，就是 1/2 的子句数量：因为每次的划分都取了子集中更大的部分，同时加起来是原始子句集，及总子句数量。所以全部加起来也一定大于 1/2 总子句数量。

ACKNOWLEDGMENT

Algorithm Design and Analysis is such a great class taught by **Prof. Dongbo Bu**[2] that I would vote for it of one of the best classes in UCAS without hesitating.

This edition of hint is lack of revising, which is just one rough collection of solving ideas. I do not guarantee any solution of problem is absolutely correct. The author of this hint thanks for predecessors and peer guys' help to finish this work. However, some references, which are just ugly copying, may be lack of authorization, thus please contact me freely if necessary. **I would be really sorry if these operations bother you and stop publishing this document at once.**

Specifically, thanks for the help of elie (elie_001@163.com), FeiYang (yangfei@ict.ac.cn), Sariel Har-Peled, Wu Huikai, HuangKai, ZhuYi, Jun Zhang, Chen Maosen, Lv Wenbin. The figure in the front cover is from http://t.sina.com.cn/1915941245.

[2] http://bioinfo.ict.ac.cn/~dbu/