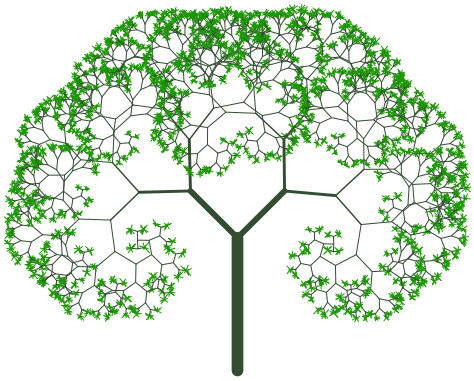
算法设计与分析课后作业

elie*

2012年2月16



*elie_001@163.com

1 小试身手——第一次作业

1.1 True or False?

True or false? Consider an instance of the Stable Matching Problem in which there exits a man m and a woman w such that m is ranked first on the preference list of w and w is ranked first on the preference list of m. Then in every stable matching S for this instance, the pair (m, w) belongs to S.

命题为真。用反证法证明。假设存在稳定匹配S,$(m, w) \in S$.那么存在一名“女性”w'和一名“男性”m',使得$(m, w') \in S, (m', w) \in S$. 又因为对于$m$,他更喜欢$w$而非$w'$,对于$w$,她更喜欢$m$而非$m'$,从而$S$是不稳定的。因此与假设$S$是稳定的矛盾。从而对做任意的一个稳定的匹配$S$,$(m, w) \in S$.

1.2 True or False?

True or false? In the stable matching S generated by G-S algorithm, each woman is paired with her worst valid partner.

命题为真。 设w_1的worst valid partner是m_1.假设G-S算法产生的稳定匹配使w_1不与m_1相配。不妨设w_1与m_2相配。那么w_1更喜欢m_2而非m_1(因为m_1是w_1的wosrt valid patner)。设另一匹配S中w_1与m_1相配。m_2与w_2相配。在S^*中m_2的配偶w_1是m_2的best valid patner.故m_2更喜欢w_1而非w_2,又w_1更喜欢m_2而非m_1.从而$w_1 - m_2$不稳定。

1.3 True or False

Suppose we have two television stations, $\mathcal{A}$ and $\mathcal{B}$. There are n prime-time programming slots, and each station has n TV shows. Each station wants to assign each show to a distinct slot so as to attract as much market share as possible. Here is the way we determine how well the two stations perform relative to each other, given their schedules. Each show has a fixed score. We'll assume that no two shows have exactly the same score. A station wins a given time slot if the show that it schedules for the time slot has a higher score than the show the other station schedules for that time slot. The goal of each station is to win as many time slots as possible.

Suppose Station $\mathcal{A}$ reveals a schedule S and Station $\mathcal{B}$ reveals a schedule T. We'll say that the pair of schedule (S, T) is stable if neither station can unilaterally change its own schedule and win more time slots. That is, there is no schedule S' such that Station $\mathcal{A}$ wins more slots with the pair (S', T) than it did with the pair (S, T); and symmetrically, there is no schedule T' such that Station $\mathcal{B}$ wins more slots with the pair (S, T). Decide whether you

think the following statement is true or false. If it is true, give an algorithm. If it is false, give a counterexample.

True or false? For every set of TV shows and scores, there is always a stable pair of schedules.

命题为假。考 $n = 2$ 的情形，设 $\mathcal{A}$ 的两个节目的分数分别为 a_1, a_2, $\mathcal{B}$ 的两个节目的分数分别为 b_1, b_2, 并且 $a_1 > b_1 > a_2 > b_2$. 以这样的分数，无论怎么样的计划，两公司总可以单方面地改变计划以获得更多的播放机会。例如，a_1, b_1 竞争第一个节目时间，a_2, b_2 竞争第二个节目时间，这样的计划 $\mathcal{B}$ 公司获得零个播出时间。但是 $\mathcal{B}$ 公司可以改变策略，让 b_1 与 a_2 竞争第二个节目，b_2 与 a_1 竞争第一个节目，这样，$\mathcal{B}$ 公司可以获得一个播出时间。同样，a_1 与 b_2 竞争，a_2 与 b_1 竞争时，$\mathcal{A}$ 以改变策略获得更多的播出时间。

1.4 Stable Matching

PSL is a shipping company that owns n ships and provide service to n ports. Each ship has a schedule that says, for each day in some $m(m > n)$ day period, whether the ship is at sea or staying at one port. During the period, each ship visits each port once, and each visit lasts exactly one day. For safety reasons, PSL has the following strict requirement:

(†) No two ships can be in the same port on the same day.

The company wants to perform maintenance on all the ships in the following m day period. In this case, each ship Si will arrive in one scheduled port and simply remains there for the rest of the period, ignoring the rest of its schedule. Now the question is: Given the schedule for each ship, find when each ship should stop its travel so that condition (†) continues to hold. Try to give an algorithm to find the solution.

此问题类似于稳定婚姻问题。只需将船类比成稳定婚姻问题中的男士，码头类比成女士，然后套用 G-S 算法。

1.5 Complexity Analysis

Design a data structure to support the following two operations for a set S of integers:

INSERT(S, x) inserts x into set S;

DELETE-LARGER-HALF(S) deletes the largest $\lfloor S/2 \rfloor$ elements from S.

Explain how to implement this data structure so that any sequence of m operations runs in $O(m)$ time.

将数据结构设计为一个动态数组 S.

1. INSERT（插入操作）

   每次向数组插入一个元素。当数组存满时，若执行插入操作，则另外申请加倍的内存空间，并将原空间的数据拷贝到新申请的空间中，删除原内存空间。

2. DELETE-LARGER-HALF（删除操作）

忽略对删除操作前数据为空的判断时间。每次删除操作分两步：

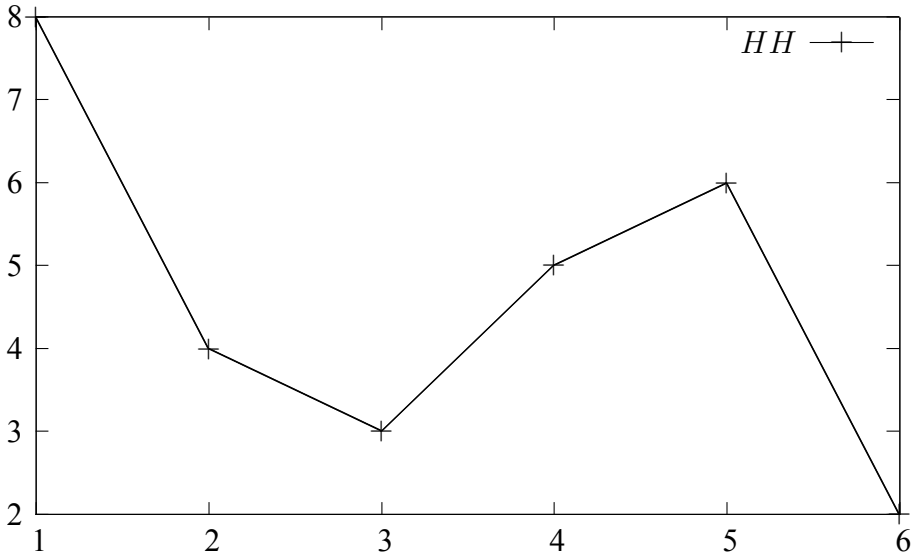
- 获取数组中所有元素的中位数（可以采用《算法导论》（第二版）*9.3* 节最坏情况线性时间获取线性表中位数的算法）。
- 删除数组中最大的 $\lfloor S/2 \rfloor$ 个元素，只需扫描整个数组，如果扫描过程中当前元素小于中位数，则添加到数组的前半部分。如果当前元素大于或等于中位数，则删除这个元素。

可以用平摊分析法，得出任何次序的插入和删除操作的时间复杂度是 $O(n)$.

1.6 Complexity Analysis

In stock market, HH-index(historically highest) of the current price is k means that current price is the highest price in the previous k days, but not the highest one in the previous $k+1$ days.

For example, the price changes as showed in the following figure.



The price in Day 5 is the highest in Days $5, 4, 3, 2$, but not highest one in Days $5, 4, 3, 2, 1$, thus $HH(5) = 4$;

The price in Day 4 is the highest in Days $4, 3, 2$, but not highest one in Days $4, 3, 2, 1$, thus $HH(4) = 3$;

The price in Day 3 is the highest in Day 3, but not highest one in Days $3, 2$, thus $HH(3) = 1$;

The input in this example is $8, 4, 3, 5, 6, 2$, the answer is $1, 1, 1, 3, 4, 1$.

Given the prices of n days, please give an algorithm of $O(n)$ time complexity to calculate the HH-index of all days.

解：只简单地给出算法(算法分析是简单的——时间复杂度$O(n)$)：

while 还有没处理的数据 **do**
  将下降序列插入到队列Q中。
  将上升序列插入到栈S中。
  $k_1 = |S|; k_2 = |Q|;$
  while $S \neq \emptyset$ **do**
    $t = s.\mathrm{pop}();$
    $HH[t] = k;$
    while $Q \neq \emptyset$ and $v[Q.\mathrm{top}()] > v[t]$ **do**
      $t_q = Q.\mathrm{pop}();$
      $HH[t_a] = 1;$
    end while
    $HH[t] += |Q| + 1;$
  end while
  while $Q \neq \emptyset$ **do**
    $t_q = Q.\mathrm{pop}();$
    $HH[t_q] = 1;$
  end while
end while

2 NP 完全问题的证明

2.1 NP-completeness

The SUBGRAPH-ISOMORPHISM problem takes two graphs G_1 and G_2 and asks whether G_1 is isomorphic to a subgraph of G_2. Show that the SUBGRAGH-ISOMORPHISM problem is NP-complete.

Proof. First we prove that the SUBGRAPH ISOMORPHISM problem is in NP. The certificate is $(G_1 = (V_1, E_1), G_2 = (V_2, E_2), \varphi V_1 \to V_2)$. The verifying algorithm checks if φ is a one-to-one function, and for all $u, v \in V_1$ whether $(u, v) \in E_1$ if and only if $(\varphi(u), \varphi(v)) \in E_2$.

Secondly, we prove that CIQUE $\leq_{\mathrm{P}}$ SUBGRAGH ISOMORPHISM. Let $(G = (V, E), k)$ be an input instance for CLIQUE. Define G_1 to be the complete graph on k vertices, and G_2 to be the grapgh G. Then $(G_1, G_2) \in$ SUBGRAGH ISOMORPHISM if and only if $(G, k) \in$ CLIQUE. □

2.2 NP-completeness

Given an integer $m \times n$ matrix A and an integer m-vector $\boldsymbol{b}$, the 0-1 INTEGER PROGRAMMING problem asks whether there is an integer n-vector $\boldsymbol{x}$ with elements in the set $\{0, 1\}$ such that $A\boldsymbol{x} \geq \boldsymbol{b}$. Prove that 0-1 INTEGER PROGRAMMING is NP-complete.

Proof. First we show that 0-1 INTEGER-PROGRAMMING is in NP. The certificate is the n-vector $\boldsymbol{x}$ with elements in the set $0, 1$. Given the certificate, it can be verified in linear time (hence, polynomial time) that it satisfies $A\boldsymbol{x} \leq \boldsymbol{b}$.

3SAT $\leq_{\mathrm{P}}$ 0-1 INTEGER-PROGRAMMING

Goal:

Next we want to show that 3SAT is polynomially reducible to the 0-1 INTEGER-PROGRAMMING (IP). That is, we want a polynomial time computable function, which given an instance of 3SAT (a set of variables Y and clauses C) outputs an instance of IP (an m-by-n matrix A and a m-vector $\boldsymbol{b}$) such that the clauses C are satisfiable if and only if there is an integer n-vector $\boldsymbol{x}$ with elements in the set $\{0, 1\}$ which satisfies $A\boldsymbol{x} \leq \boldsymbol{b}$.

Transformation:

For each clause C we write out an inequality as follows. If the literal is variable y_i, we replace it by the variable x_i; and if the literal is $\bar{y_i}$, we replace it by the quantity $1 - x_i$. In what follows, we will call this quantity that replaces the literal a term. Finally, the sum of these terms is set ≥ 1. For example, if the 3SAT instance consists of two clauses $y_1 \vee \bar{y_2} \vee \bar{y_3}$ and $\bar{y_1} \vee y_3 \vee y_4$, then the inequalities are $x_1 + (1 - x_2) + (1 - x_3) \geq 1$ and $(1 - x_1) + x_3 + x_4 \geq 1$. One can obtain the matrix A and vector $\boldsymbol{b}$ from these inequalities quite easily. The entire

transformation can be done in linear time (hence polynomial time).

Correctness:

If the clauses are satisfiable then consider the satisfying truth assignment. For each variable y_i, if y_i is true, then the corresponding variable x_i is assigned value 1, and if y_i is false, then the corresponding variable xi is assigned value 0. Consider any clause $c \in C$. Clearly, the satisfying truth assignment makes at least one literal in the clause c true. If this literal is the variable y_i then x_i has value 1 and the inequality corresponding to clause c is satisfied; and if this literal is y_i then x_i has value 0, and so $1 - x_i$ has value 1 and again the inequality corresponding to clause c is satisfied. Since the argument applies to any clause c, it implies that all the inequalities are satisfied by the above assignment. Conversely, if all the inequalities can be satisfied then consider the assignment of values to the variables x_i that satisfies the inequalities. For each variable x_i, if x_i is 1, then the corresponding variable y_i is set to true, and if x_i is 0, then the corresponding variable y_i is set to false. Consider any inequality and the corresponding clause $c \in C$. Clearly, if the inequality is satisfied then one of the three terms must be ≥ 1 (since all the terms are ≥ 0). If the term is of the form x_i then it implies that x_i is 1 and the corresponding literal y_i is true; and if the term is of the form $1 - x_i$ then it implies that x_i is 0 and the corresponding literal y_i is true. In either case, the clause c is satisfied. Our argument applies to any arbitrary clause c, hence the given clauses are satisfiable. □

2.3 NP-completeness

The SET-PARTITION problem takes as input a set S of numbers. The question is whether the numbers can be partitioned into two sets A and B and $B = S - A$ such that $\Sigma_{x \in A} x = \Sigma_{x \in B} x$. Show that the set-partition problem is NP-complete.

Proof. First we show that SET-PARTITION is in NP. The certificate for the SET-PARTITION problem consists of the two sublists S_1 and S_2. Given the sublists, in polynomial time we can compute the sums of the elements in these lists and verify that they are equal. Subset SUM $\leq_{\mathrm{P}}$ SET-PARTITION

Goal:

Next we want to show that SUBSETSUM (SS) is polynomially reducible to the SET-PARTITION problem (PART). That is, we want a polynomial time computable function f, which given an instance of SS (a set of numbers $S = \{x_1, x_2, \ldots, x_n\}$ and a target value t) outputs an instance of PART (a set of numbers $S' = \{x'_1, x'_2, \ldots, x'_n\}$) such that S has a subset summing to t if and only if S' can be partitioned into subsets S_1 and S_2 that sum to the same value.

Transformation: Observe that the SET-PARTITION problem is a special case of the subset sum problem where we are trying to find a set of numbers S_1 that sum to half the total sum of the

whole set. Let T be the sum of all the numbers in S.

$$T = \sum_{i=1}^{n} x_i.$$

If $t = T/2$ then the subset sum problem is an instance of the partition problem, and we are done. If not, then the reduction will create a new number, which if added to any subset that sums to t, will now cause that set to sum to half the elements of the total. The problem is that when we add this new element, we change the total as well, so this must be done carefully.

We may assume that $t \leq T/2$, since otherwise the subset sum problem is equivalent to searching for a subset of size $T - t$, and then taking the complement. Create a new element $x_0 = T - 2t$, and call partition on this modified set. Let S' be S together with this new element: $S' = S \cup \{x\}$. Clearly the transformation can be done in polynomial time.
Correctness: To see why this works, observe that the sum of elements in S' is $T + T - 2t = 2(T - t)$. If there is a solution to the subset sum problem, then by adding in the element x_0 we get a collection of elements that sums to $t + (T - 2t) = T - t$, but this is one half of the total $2(T - t)$, and hence is a solution to the SET-PARTITION problem. Conversely, if there is a solution to this SET-PARTITION problem, then one of the halves of the partition contains the element x0, and the remaining elements in this half of the partition must sum to $(T - t) - (T - 2t) = t$. Thus these elements (without x_0) form a solution to the subset sum problem. □

2.4 NP-completeness

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 a truth assignment to the variables of ϕ such that exactly half the clauses evaluate to 0 and exactly half the clauses evaluate to 1. Prove that the HALF-3SATproblem is NP-complete.

Proof. In order to prove that the HALF-3SAT problem is NP-Complete, we need to prove two things:

1. HALF 3SAT $\in$ NP,
2. $\forall \mathrm{L} \in \mathrm{NP}, \mathrm{L} \leq_{\mathrm{P}}$ HALF-3SAT, or HALF-3SAT $\in$ NP-Hard

In order to prove that HALF-3SAT is in NP, we need to show that a solution to a particular instance of the problem can be verified in polynomial time. Consider the following ”witness” algorithm which takes an instance of HALF-3SAT and a ”certificate” as parameters. Our algorithm takes a boolean formula in conjunctive normal form, and the ”certificate” or ”proposed solution” is a list of the boolean variables and their proposed truth assignments.

The algorithm walks the formula, evaluating each clause with the proposed truth assignments and returns true if exactly half the clauses are true, and false if not. This is clearly polynomial time because the boolean formula is only walked once.

In order to prove that HALF-3SAT is NP-Hard, we can consider a reduction from a known NP-Complete problem to HALF-3SAT (or prove it directly, which we won't even consider). We choose to reduce from 3SAT to HALF-3SAT. Because CIRCUIT-SAT $\leq_{\mathrm{P}}$ SAT $\leq_{\mathrm{P}}$ 3SAT, if we are able to show 3SAT $\leq_{\mathrm{P}}$ HALF 3SAT, then this implies that CIRCUIT-SAT $\leq_{\mathrm{P}}$ HALF 3-SAT. And since L $\leq_{\mathrm{P}}$ CIRCUIT-SAT, $\forall$L $\in$ NP, then this implies that L $\leq_{\mathrm{P}}$ HALF 3SAT, $\forall$L $\in$ NP or that HALF 3SAT $\in$ NP-hard.

Consider the following reduction. First, we need to convert an instance of 3SAT to HALF 3SAT. Our approach is create a φ' which contains 4 times as many clauses as φ. Suppose φ contains m clauses. When creating φ', first we take all of the clauses from φ. Next, we create m clauses of the form:

$$(p \vee \neg p \vee q)$$

Clearly, these clauses are always true, regardless of the individual truth assignments for p and q. Next, we create $2m$ clauses of the form:

$$(p \vee q \vee r)$$

These clauses are always true or always false. Therefore, we have created a boolean formula φ' which contains all of the clauses φ and m clauses which are always true and $2m$ clauses which are either all true or all false. Clearly, this conversion takes polynomial time because we have only added 3 variables, and $3m$ clauses.

We also need to show that there exists a "yes" instance of 3SAT if and only if there exists a "yes" instance of HALF 3SAT.

- $\Longrightarrow$ that there exists a truth assignment which causes φ to be true. Then, the m clauses which correspond to φ in φ' are true and there are m clauses which are always true. Thus, simply let p and r be false, and there exists a truth assignment which satisfies HALF 3SAT, where half the clauses are true and half are false.

- $\Longleftarrow$ that there exists a truth assignment which causes HALF 3SAT to be satisfied, or a truth assignment that causes half the clauses in φ' to be true and half false. But m clauses in φ' are always true, which means that the $2m$ clauses cannot be true if HALF 3SAT is satisfied (because then $3m$ clauses would be true which is more than half). Thus, the $2m$ clauses must be false, which means that φ is true, which means that 3SAT is also satisfied.

Thus, we have shown that a "yes" instance of 3SAT produces a "yes" instance of HALF 3SAT, and vice versa, which concludes our proof. $\square$

2.5 NP-completeness

Show that UNDIRECTED-HAMILTON-CYCLE problem is NP-complete.

UNDIRECTED-HAMILTON-CYCLE(DECISION PROBLEM):

Input: an undirected gragh G

Output: 1 if G has a Hamilton cycle which visits each vertex exactly once and also returns to the starting vertex, 0 for others.

证明：该问题显然是NP问题，故只需证明该问题是NP-Hard问题。记Hamilton轨问题为HP, Hamilton圈问题为HC, 有向Hamilton圈问题为DHC, 有向Hamilton轨问题为DHP.

首先证明HP $\leq_{\mathrm{P}}$ HC. 是实上, 若HP的输入为I: 一个无向图G和$u_0, v_0 \in V(G)$, 则取HC的输入为$f(I) : G' = G + v_0u_0$. 这样在G中存在Hamilton轨$P(u_0, v_0)$当且仅当G'存在Hamilton圈。

其次证明DHC $\leq_{\mathrm{P}}$ DHP和（非常简单, 类似上面的证明，略）

最后只需证明DHP $\leq_{\mathrm{P}}$ HP. 令有向图G'和两顶$v_a, v_b \in V(G')$, 是DHP的一个输入, 我们由G'来构造一个无向图G如下：对每个顶$v_i \in V(G')$, G含三个顶点v_i^1, v_i^2, v_i^3, 和两条边$v_i^1v_i^2, v_i^2v_i^3$. 对每一条有向边$v_iv_j \in E(G')$, G中含边$v_i^3v_j^1$. $f(I)$取G和v_a^1, v_b^3作为对应的输入。我们往证G有v_a^1与v_b^3之间的Hamilton轨的充分必要条件是G'有从v_a到v_b的有向Hamilton轨。若G'有从v_a到v_b的一条有向Hamilton轨，则G有在v_a^1到v_b^3之间的Hamilton轨是显然的。反之，设$P(v_a^1, v_b^3)$是G中的一条Hamilton轨，我们从v_a^1开始沿$P(v_a^1, v_b^3)$运行，把$P(v_a^1, v_b^3)$上形如$v_i^1v_i^2v_i^3$的长2的子轨缩成一个顶点v_i, 从而得到G'上从v_a到v_b的一打有向Hamilton轨。

综上有DHC $\leq_{\mathrm{P}}$ DHP $\leq_{\mathrm{P}}$ HP $\leq_{\mathrm{P}}$ HC, 而DHC是NPC问题，故HC是NPC问题。□

2.6 NP-completeness

In the HALF-CLIQUE problem, we are given a graph G with n nodes, where n is even. We wish to determine whether there exists a subgragh of G with $n/2$ nodes that is a clique. Show that the HALF-CLIQUE problem is NP-complete.

证明：

1. 半团问题(HALF-CLIQUE)是NP问题。
   证书是一个子图H和与图G的顶点的映射关系f, 只需验证H是完全图，其顶点个数为$n/2$, 且$\forall v_1, v_2 \in V(H), (f(v_1), f(v_2)) \in E(G)$. 这显然可在多项式时间内完成。

2. CLIQUE $\leq_{\mathrm{P}}$HALF-CLIQUE. 设$\langle G_1, k\rangle$是一个团的实例。

   - $k \geq |G|/2$
     在G上添加$j = 2k - |G|$个孤立顶点，得到图G'. G'中含有半团，当且公当

原图G中含有k个顶点的团。

- $k < |G|/2$
  在G上添加$m = |G| - 2k$个顶点，并且与G中的每个顶点完全连接，形成图G'. 现在G'具有$2|G| - 2k$个顶点，且G'具有规模为$|G| - k$的半团当且仅当原图G中含有规模为$k + |G| - 2k = |G| - k$的团。

3 分治法

3.1 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: 设第一个数据库的第 k 个元素为 a_k, 第二个数据库的第 k 个元素为 b_k. 令

$$f(k) = a_k - b_{n-k+1}$$

. 那么 $f(k)$ 是关于 k 的单调上升函数且如果 $f(1) \geq 0$ 即 $a_1 \geq b_n$ 则这两个数据库的第 n 大的元素就为 b_n. 同理, 若 $f(n) \leq 0$, 则第 n 大的元素为 a_n. 否则存在一个正整数 k 使得

$$f(k) \leq 0, f(k+1) \geq 0.$$

从而有

$$a_1 \leq a_2 \leq \ldots a_k \leq b_{n-k+1},$$
$$b_1 \leq b_2 \leq \ldots b_{n-k} \leq a_{k+1}.$$

从而存在第一个数据库的元素 a_{k+1} 和第二个数据库中的元素 b_{n-k+1} 都大于集合 $A = \{a_1, a_2, \ldots, a_k, b_1, b_2, \ldots, b_{n-k}\}$ 中的任何一个元素且 $|A| = n$. 易证两数据库中第 n 大的元素一定在集合 A 中, 从而必是 $\max A = \max\{a_k, b_{n-k}\}$. 由此可根据二分法的思想搜索出这样的 k, 就找到了第 n 大的元素了。

FindNthSmallest($a[1..n], b[1..]$)

```
i = 1, j = n;
if a[1] ≥ b[n] then
  return b[n];
end if
if a[n] ≤ b[1] then
  return a[n];
end if
while j − i > 1 do
```

$k = \lfloor \frac{j+i}{2} \rfloor$;
$temp = a[k]$;
$t = temp - b[n-k+1]$;
if $t > 0$ **then**
$j = k$;
else if $t < 0$ **then**
$i = k$;
else
return $temp$;
end if
end while
return $\max\{a[k], b[n-k]\}$;

3.2 Divide and Conquer

Recall the problem of finding the number of inversions. As in the course, we are given a sequence of n numbers $a_1, \ldots, a_n$, 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 $a_i > 2a_j$. Given an $O(n\log n)$ algorithm to count the number of significant inversions between two orderings.

Sort-and-Count (A)

1: Divide A into two sub-sequence L and R;
2: (RC_L, L) = Sort-and-Count(L);
3: (RC_R, R) = Sort-and-Count(R);
4: (r, A) = Merge-and-Count(L, R);
5: **return** $(RC = RC_L + RC_R + r, A)$;

Merge-and-Count (L, R)

1: $InverseCount = 0$;
2: $i = 1; j = 1$;
3: **for** $k = l$ to r **do**
4: **if** $L[i] > R[j]$ **then**
5: $A[k] = L[j]$;
6: $j++$;
7: **else**
8: $A[k] = R[i]$;

```
    i++;
  end if
end for
i = 1, j = 1;
for k = Ltor do
  if L[i] > 3 · R[j] then
    InverseCount+ = length(L) − i + 1;
    ++j;
  else
    ++i;
  end if
end for
return InverseCount and A;
```

3.3 Divide and Conquer

Consider an n-mode 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:算法如下

FindLocalMin(T)

```
v_l = probe(T.leftchild());
v_r = probe(T.rightchild());
v = probe(T.root())
if v < v_1 then
  if v < v_r then
    return T.root();
  else
    FindLocalMin(T.rightchild());
  end if
else
  if v_l < v_r then
```

```
12:      FindLocalMin(T.leftchild());
13:    else
14:      FindLocalMin(T.rightchild());
15:    end if
16: end if
```

算法的基本思想是，找到根节点与根节点的两个孩子节点的值最小的，如何最小者恰好是根节点，那么就返回该节点，否则递归查找孩子中值较小的那一个对应的子树。算法中，probe() 接受一个树的节点的输入，返回该节点的值。

3.4 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 $1 \leq i \leq n$ and $1 \leq j \leq n$; 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 the previous question. 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。

3.5 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：剖分的个数是 Catalan 数，组合数学上有详细的介绍，这里就不再给出。

3.6 Divide and Conquer

Suppose you have n (n is even) coins, and you have a device to decide whether two coins are made of the same material. (You cannot use the device to compare two piles of coins.)

Now you want to decide whether there is a set of more than $n/2$ of these coins that are made of the same material.

(a) Give an algorithm using $O(n \log n)$ time to solve this problem.

(b) Give an algorithm using $O(n)$ time and $O(1)$ space to solve this problem.

sorry! I have an algorithm but fail to give a proof. So I'm not sure whether the algorithm is correct. Under this circumstance, I decide not to give the solution of this problem and am hoping someone else will give a satisfying answer. I'm looking forward for your answer.

4 动态规划

4.1 Dynamic Programming

Professor Stewart is consulting for the president of a corporation that is planning a company party. The company has a hierarchical structure; that is, the supervisor relation forms a tree rooted at the president. The personnel office has ranked each employee with a conviviality rating, which is a real number. In order to make the party fun for all attendees, the president does not want both an employee and his or her immediate supervisor to attend.

Professor Stewart is given the tree that describes the structure of the corporation, using the left-child, right-sibling representation. Each node of the tree holds, in addition to the pointers, the name of an employee and that employee's conviviality ranking. Describe an algorithm to make up a guest list that maximizes the sum of the conviviality ratings of the guests. Analyze the running time of your algorithm.

解:只给出递推关系式:

$$f_1(m) = \mathrm{conv}(m) + \sum_{p \in \mathrm{post}(m)} f_(p),$$
$$f_0(m) = 0 + \sum_{p \in \mathrm{post}(m)} \max\{f_1(p), f_0(p)\}$$

其中 $\mathrm{conv}(m)$ 表示雇员 m 的 Conviviality rating, $\mathrm{post}(m)$ 表示 m 的直接下级组成的集合。所以要求的就是 $\max f_0(m_0), f_1(m_0)$ 以及取得最大值的方案,这里 m_0 表示"顶头上司"。

时间复杂度为 $O(n)$,因为每个雇员被计算两个值 f_1, f_2,考虑计算 f_1 的计算量(假设 f_0 已经计算出),当所有雇员的 f_1 值都被计算后,所用的加法的次数为 $2n-1$,考虑计算 f_0 的计算量,它需要 $n-1$ 次比较和少于 $n-1$ 次的加法,因此,时间复杂度为 $O(n)$.

4.2 Dynamic Programming

Suppose you have one machine and a set of n jobs $a_1, a_2, ..., a_n$ to process on that machine. Each job a_j has a processing time t_j, a profit p_j, and a deadline d_j. The machine can process only one job at a time, and job a_j must run uninterruptedly for t_j consecutive time units. If job a_j is completed by its deadline d_j, you receive a profit p_j, but if it is completed after its deadline, you receive a profit of 0. Give an algorithm to find the schedule that obtains the maximum amount of profit, assuming that all processing times are integers between 1 and n. What is the running time of your algorithm?

Solution:

Recursion Relation: We first order the jobs by increasing deadline so $a_1 \ldots, a_n$ are or-

dered by deadline. We then recurse on $P(i, j)$, the maximum profit we can make using jobs $1, 2, \ldots, i$ in time j. The relation is:

$$P(i,j) = \begin{cases} 0, & \text{if } i = 0, \\ \max(P(i-1,j), P(i-1,j-t_i) + p_i), & \text{if } j \leq d_i, \\ P(i-1,j), & \text{else.} \end{cases}$$

Running Time: Since all processing times are no more than n, we need only compute j to n^2. Therefore, P is size $O(n^3)$. Filling in each square of P requires $O(1)$ time for a total running time of $O(n^3)$. Note that the $O(n \log n)$ sorting cost has been absorbed.

4.3 Dynamic Programming

To assess how "well-connected" two nodes in a directed graph are, one can not only look at the length of the shortest path between them, but can also count th $number$ of shortest paths.

This turns out to be a problem that be solved efficiently, subject to some restrictions on the edge costs. Suppose we are given a directed graph $G = (V,\ E)$, with costs on the edges; the costs may be positive or negative, but every cycle in the graph has strictly positive cost. We are also given two nodes $v,\ w \in V$. Give an efficient algorithm that computes the number of shortest $v - w$ paths in G.

Solution:

First we apply the Fullman-Fulkson algorithm to get a shortest path P and the length of the path. We assume that the shortest path of the graph G is k. For every edge e of the path P, we get a graph $G_e = G - e$. We apply the Fulkson-Fulkson algorithm on every of these graphs and get the shortest path's length, denoted as k_e. If there is a $k_e = k$, for some e in P, the shortest path is not uniqe, otherwise, it is uniqe.

4.4 Dynamic Programming

You‘re running a computing system capable of processing several terabytes of data per day. For each of n days, you're presented with x_i terabytes on day i. Note that any unprocessed data is abandoned at the end of the day.

However, the computing system can only process a fixed number of terabytes in a given day. Besides, the amount of data you can process goes down every day since the most recent reboot of the system. On the first day after a reboot, you can process s_1 terabytes, and so on, up to s_n; we assume $s_1 > s_2 > s_3 > \cdots > s_n > 0$. To get the system back to peak performance, you can choose to reboot it; but on any day you choose to reboot the system, you can't process any data at all.

Now please give an efficient algorithm that takes values for $x_1, x_2, \ldots, x_n$ and $s_1, s_2, ..., s_n$ and returns the total number of terabytes processed by an optimal solution.

解:(解法一)假设第0天和第$n+1$天重启了系统,这并不影响条件。$f[i,j]$表示第$i-1$天和第$j+1$天重启系统的情况下,第i天到第j天处理的数据的最优值。那么$f[1,n]$就是要求的最优值,$f[1,n]$对应的方案就是最优方案。有如下的递推关系:

$$f[i,j]=\begin{cases}\min\{s_1,x_i\}, & i=j,\\ 0, & j<i,\\ \max_{i\le k\le j}\{f[i,k-1]+f[k+1,j],g[i,j]\}, & \text{else.}\end{cases}$$

其中$g[i,j]$表示第$i-1$天重启了系统但第i到第j天都不重启系统折情况下第i到第j天处理的数据量。

容易计算出,按照上面的弟推关系的动态规划算法的时间复杂度是$O(n^3)$.因为还有更好的算法,这里就不再仔细推导了。

(解法二)我们用$F(i,j)$表示问题的解(能处理的最大数据量),意思是还剩下i天并且我们在第j天重启了系统。假设我们总共有n天。第1天,我们有$F(n,1)$;最后一天结束时,我们有$F(0,j)=0$(对于所有j)。当没有剩余时间时,也就没有可能处理数据。

那么我们可以将问题看作是做一组决策:当还剩i天时,我们是否应该重启系统。这两种决策的情形如下:

$$F(i,j)=\max\{F(i-1,j+1)+\min\{x_{n-i+1},s_j\},F(i-1,1)+0\}$$

需要注意的是:当还剩下i天时,需要处理的数据是x_{n-i+1},而系统能处理的最大数据量是s_j.我们只能能其中较小的那一个。

这个算法初始化的$F(0,j)=0,\forall j\in\mathbf{N}_+$,并计算所有$i$的$F(i,j)$值。其中$j\le n-i+1$.当我们算到第一天的$F(n,1)$时,问题就解决了。

每一步需要耗费$O(1)$的时间,并且共有$O(n^2)$步对应于所有的i和j.因此,算法的时间复杂度是$O(n^2)$.

4.5 Dynamic Programming

Given a sequence of n real numbers $a_1, ..., a_n$, determine a subsequence (not necessarily contiguous) of maximum length in which the values in the subsequence form a strictly increasing sequence.

解:假设我们知道了以字符a_i结尾的子串长度C_i,因此对所有k,有:

$$C_k=1+\max_{i\in\{i|0\le i<k,a_i<a_k\}}C_i.$$

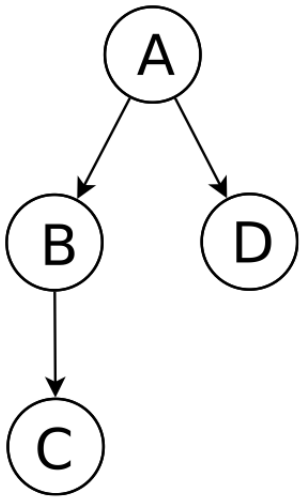
根据这个公式，我们可以得到以$a_1, a_2, \ldots, a_n$结尾的所有子串长度。最后选择其中最大的一个结果返回。

容易得到，算法时间复杂度为$O(n^2)$.

4.6 Dynamic Programming

In an army system, to notify everyone of the postponement, the ranking officer first calls each of his/her direct subordinates, one at a time. As soon as each subordinate gets the phone call, he/she must notify each of his/her direct subordinates, one at a time. The process continues this way until everyone has been notified. Note that each person in this process can only call direct subordinates on the phone.

We can picture this process as being divided into rounds. In one round, each person who has already learned of the postponement can call one of his/her direct subordinates on the phone. The number of rounds it takes for everyone to be notified depends on the sequence in which each person calls their direct subordinate. In the following example, it will take only two rounds if A starts by calling B, but it will take three rounds if A starts by calling D.



Give an algorithm that determines the minimum number of rounds needed for everyone to be notified, and outputs a sequence of phone calls that achieves this minimum number of rounds.

解：用$r(a)$表示官员a的所有下级被通知到需要的最少的轮数。首先将没有下级的官员的r值初始化为0.我们从下级的官员到上级的官员的方向进行计算。计算官员x时，他的所有的下级官员的r值都已经被计算出来了。容易证明，官员x按他的下级r值的递减的顺序通知时，所花的轮数最少。设x的下级的r值分别为$n_1 \geq n_2 \geq \ldots, n_k$,那么$r(x) = \max_{1 \leq j \leq k}\{n_j + j\}$. 这样，当计算到首领时，算法终止。

5 贪心算法

5.1 True or False

Decide whether you think the following statement is true or false. If it is true, give a short explanation. If it is false, give a counterexample.

True or false? Let G be an arbitrary connected, undirected graph with a distinct cost $c(e)$ on every edge e. Suppose e is the cheapest edge in G; that is, $c(e) < c(e)$ for every edge $e \neq e$. Then there is a minimum spanning tree T of G that contains the edge e.

命题为真 运用反证法证明。假设存在一棵不包含 e^* 的最小生成树 T, 使得树的所有边的权的总和 $c(T)$ 最小。可以如下构造一棵新树：将 e^*（设 e^* 相关联的两顶点为 A, B）添加到 T 上，则在 A, B 之间存在两条不同的路径相互连通。一条在碑 T 的内部，（由最小生成树的定义，T 是连通的），一条就是 e^*. 即形成了一个包含 A, B 的圈。我们可以选择去掉这个圈内的权值最大的边，形成一棵连通所有顶点的新树 T'，并且 $c(T') < c(T)$. 这与 T 是最小生成树的假设矛盾。因此，最小生成树必然包含边 e^*.

5.2 Greedy Algorithm

Let us say that a graph $G = (V, E)$ is a *near-tree* if it is connected and has at most $n + 8$ edges, where $n = |V|$. Give an algorithm with running time $O(n)$ that takes a near-tree G with costs on its edges, and returns a minimum spanning tree of G. You may assume that all the edge costs are distinct.

解：当 G 不是树时，如下迭代：

- 用尝试优先搜索或广度优先搜索找到 G 中的一个环。
- 删除此环中最大的边

迭代 8+1 次，留下正好含 $n - 1$ 条边的树。每次迭代用结点的线性时间（每个结点最多搜索、检查常数次）。因此时间复杂度是 $O(n)$.

5.3 Greedy Algorithm

Given a list of n natural numbers d_1, d_2,...,d_n, show how to decide in polynomial time whether there exits an undirected graph $G = (V, E)$ whose node degrees are precisely the numbers d_1, d_2,...,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：算法如下：isGraphsDegrees($d[1..n]$)

1: $p = \sum_{i=1}^{n} d[i]$;
2: **if** $p > n(n-1)/2$ or p is odd **then**

```
    return FALSE;
  end if
  Sort d[1..n] in ascending order;
  while n > 1 and d[n] ≠ 0 do
    k = n − d[n];
    if k < 1 then
      return TRUE
    end if
    for i = k to n − 1 do
      d[i] = d[i − 1];
      if d[i] < 0 then
        return FALSE;
      end if
    end for
    n = n − 1;
    if k > 2 and d[k] < d[k − 1] then
      d = merge(d[1..k − 1], d[k..n]);
    end if
  end while
  if d[1] ≠ 0 then
    return FALSE;
  else
    return TRUE;
  end if
```

程序的正确性证明：

存在一个图G的顶点的度按升序排序为$d_1, d_2, \ldots, d_n$当且仅当存在一个图G',它的的度依次为$d_1, d_2, \ldots, d_k - 1, d_{k+1} - 1, \ldots, d_{n-1} - 1$,其中$k - n - d_n$.

证明：$\Leftarrow$)若存在图G',它的顶v_i的度为

$$d'_i = \begin{cases} d_i, & 1 \le i \le k; \\ d_i - 1, & k \le i \le n-1. \end{cases}$$

其中$k = n - d_n$.那么增加一个顶点v_n,并从v_n引边$v_nv_k, v_nv_{k+1}, \ldots, v_nv_{n-1}$(共$d_n$条边)。从而所得的图的度依次为$d(v_i) = d_i,\ i = 1, 2, \ldots, n$.

$\Rightarrow$)设图G的各顶满足$d(v_i) = d_i, i = 1, 2, \ldots, n$. $d_1 < d_2 < \cdots < d_n$.若与v_n相邻的顶点就是$v_k, v_{k+1}, \ldots, v_{n-1}$,那么命题得证。否则,首先将一端在$\{v_k, v_{k+1}, \ldots, v_{n-1}\}$,另为$v_n$的所有边删除,得到图$G_1$.

记$A = \{v \in \{v_k, v_{k+1}, \ldots, v_{n-1}\} | v$不是$v_n$的邻点$\}$.对$A$中的每个元素$v, d(v) \ge 1$.

故可取一邻点 v', 删除边 vv', 对 v_n 的一边 $vv'' \in E(G_1)$, 删除边 v_nv'', 增加边 $v'v''$. 当对 A 中每个元素都执行了以上操作之后, 记所得的图为 G_1'. 那么

$$\begin{aligned} d_{G_1'}(v_n) &= 0; (\text{因为 } |A| = E(G_1)) \\ d_{G_1'}(v) &= d_G(v) - 1, \forall v \in \{v_k, v_{k+1}, \ldots, v_{n-1}\}; \\ d_{G_1'}(v) &= d_G(v), \forall v \in \{v_1, v_2, \ldots, v_{k-1}\}; \end{aligned}$$

令 $G' = G_1' - v_n$. G' 就是满足要求的图。

算法就是执行这样的图消减的过程: 每次 **while** 循环都删除了度最大的顶点。按照上面的命题, 在执行一步 **while** 循环前的图的各顶的度为 $d_1, d_2, \ldots, d_n$, 那么一次 **while** 循环后的图的度为 $d_1, d_2, \ldots, d_k - 1, d_{k+1} - 1, \ldots, d_{n-1} - 1$. 如果 $d_1, d_2, \ldots, d_{n-1} - 1$ 不能构成一个图的度的序列, 那么 $d_1, d_2, \ldots, d_n$ 也不能构成一个图的度的序列。

算法对度序列的长度进行消减, 并保证原始度序列能够生成一个图当且仅当消减后的度序列能生成一个图。因此当 **while** 循环结束时, $d[1] = 0$ 当且公当原始度序列能生成一个图。($d[1] = 0$ 表示一个单顶点构成的图 $G'' = (\{v\}, \emptyset)$)

算法的时间复杂度:

1. 计算 $\sum_{i=1}^{n} d_i$ 需要 $n - 1$ 次加法。
2. 对 $d_1, d_2, \ldots, d_n$ 排序, 需要 $O(n \log n)$ 的代价。
3. **while** 循环至多执行 $n - 1$ 次, 第 k 次 **while** 循环中 **for** 循环的次数为前 $k - 1$ 次消减过程执行后剩下的度序列中值最大的度的次数。这意味着 **for** 循环体语句执行的次数一定小于 $\sum_{i=1}^{n} d_i \leq 2n(n-1)/2 = O(n^2)$.(当 $\sum_{i=1}^{n} d_i \geq n(n-1)$ 时程序已经返回 FALSE 并退出)。另外第 k 次 **while** 循环还进行了一次合并操作, 合并了 $n - k$ 个元素, 帮总共的合并代价为 $O(\sum_{k=1}^{n-1}(n-k)) = O(n^2)$.

综上三条, 有总的时间复杂度为 $O(n^2)$.

另一方面, 事实上每执行一次 **for** 循环相当于从图中删除一边。对 n 阶完全图 K_n 共有 $n(n-1)/2$ 条边, 故总的时间代价为 $\Omega(n^2)$. 于是有算法的时间复杂度为 $\Theta(n^2)$.

5.4 Greedy Algorithm

Suppose you have n video streams that need to be sent, one after another, over a communication link. Stream i consists of a total of b_i bits that need to be sent, at a constant rate, over a period of t_i seconds. You cannot send two streams at the same time. Besides, there cannot be any delays between the end of one stream and the start of the next. Suppose your schedule starts at time 0. We assume that all the values b_i and t_i are positive integers.

Now, the link imposes the following constraint, using a fixed parameter r:

(*) For each natural number $t > 0$, the total number of bits you send over the time interval from 0 to t cannot exceed rt.

Note that this constraint is only imposed for time intervals that start at 0, *not* for time intervals that start at any other value.

Give an algorithm that takes a set of n streams, each specified by its number of bits bi and its time duration t_i, as well as the link parameter r, and determines whether there exists a valid schedule. The running time of your algorithm should be polynomial in n.

解:按b_i/t_i升序排列，并就按此序列发送。下面证明按这样的发送顺序可行。

不妨假设$b_1/t_1, b_2/t_2, \ldots, b_n/t_n$是单调增的。令$t_0 = 0$, $T_i = \sum_{j=0}^{i} t_j$; $f(t) = b_i/t_i$, 当$t \in [T_{i-1}, T_i)$; $F(t) = \int_0^t f(\xi)\mathrm{d}\xi$. 显然$F(t)$表示从0到$t$时间段发送的数据。由$f(t)$单增，$F(t)$是下凸函数。故$\forall t \in [0, T_n]$有

$$F(t) \le \frac{\sum_{i=0}^{n} b_i}{\sum_{i=1}^{n} t_i} t = \frac{\int_0^{T_n} f(t)\mathrm{d}t}{T_n} t$$

只需$\sum_{i=0}^{n} b_i / \sum_{i=1}^{n} \le r$则$\forall t \in [0, T_n]$都不会超过$rt$的限制。(另一方面，如果$\sum_{i=0}^{n} b_i / \sum_{i=1}^{n} \ge r$, 则最后一点必超过$rt$的限制。)

5.5 Greedy Algorithm

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:

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

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, $O(n \log n)$.

5.6 Greedy Algorithm

The input to this problem consists of an ordered list of n words. The length of the ith word is w_i, that is the ith word takes up w_i spaces. The goal is to break this ordered list of words into lines, this is called a layout. Note that you can not reorder the words. The length of a line is the sum of lengths of the words on that line. The ideal line length is L. No line may be longer than L, although it may be shorter. The penalty of having a line of length K is $L - K$. The total penalty is the **sum** of the line penalties. The problem is to find the layout that minimizes the total penalty.

解：对每一个词 (word):

- 如果该词能加入当前行，则将该词加入到当前行末尾；
- 如果当前行无法容纳该词，则另起一新行，并添加这个词到该新行。

假设贪心算法输出结果为G. 若对于G中第i行（任意i），将此行末尾的词放入第$i+1$行开头，

- 如果第$i+1$行能容纳这个词，可得总惩罚值并不会减少。
- 如果第$i+1$不能容纳这个词，则将第$i+1$行的末尾超出容量的几个词放入第$i+2$行的开头，不断重复这样的操作，直到某行能容纳上一行的超出的词。可得总惩罚值也不会减少。

因此证明了使用贪心算法的正确性。

算法时间复杂度：$O(n)$,n为词数。

6 线性规划

6.1 Prime and Dual

Suppose that we are given a linear program L in standard form, and suppose that for both L and the dual of L, the basic solutions associated with the initial slack forms are feasible. Show that the optimal objective value of L is 0.

解：问题的标准型表示：

$$\begin{aligned}\max f &= \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x},\\ \text{s.t.} A\boldsymbol{x} &\le \boldsymbol{b};\end{aligned}$$

化为松弛型（I 是单位阵，$\boldsymbol{s}$ 是松弛变量）：

$$\begin{aligned}\max f &= \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} + \mathbf{0}\boldsymbol{s}\\ \text{s.t.} \begin{pmatrix} A & I \end{pmatrix} &\begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{s} \end{pmatrix} = \boldsymbol{b}.\end{aligned}$$

初始松弛型具有可行解意味着 $\boldsymbol{s} = \boldsymbol{b}, \boldsymbol{x} = \mathbf{0}$, 对应 $f = \boldsymbol{c}^{\mathbf{T}}\boldsymbol{x} = 0$. 当然也是标准型的一个解。

原问题对应的对偶问题可以表示为：

$$\begin{aligned}\min f' &= \boldsymbol{b}^{\mathrm{T}}\boldsymbol{y},\\ \text{s.t.} A^{\mathrm{T}}\boldsymbol{y} &\ge \boldsymbol{c};\end{aligned}$$

$\boldsymbol{y} = \mathbf{0}$ 是对偶问题的解，此时 $f' = \boldsymbol{b}^{\mathbf{T}}\boldsymbol{x} = \mathbf{0}$.

由对偶性质知当 $f' = f$ 时，原问题和对偶问题同时达到最优解。故最优解 $f^* = 0$.

6.2 True or False

Decide whether you think the following statement is true or false. If it is true, give a short explanation. If it is false, give a counterexample.

True or false? Suppose that if we allow strict inequalities in a linear programming, and the linear programming has optimal solution, then there still exists a vertex which takes the optimal value.

解：命题正确

设最优解为 $\boldsymbol{x}^* = (x_1^*, x_2^*, \ldots, x_n^*)$（是基可行解）。不妨假设 $a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n < b_i$ 是该（扩展的）线性规划问题的一个严格不等约束，则 $a_{i1}x_1^* + a_{i2}x_2^* + \cdots + a_{in}x_n^* < b_i$ 成立. 从而将该不等约束置换为 $a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n \le b_i$, $\boldsymbol{x}^*$ 仍满足该不等式的约束。将所有这样的严格不等式约束都置换为对应的非严格不等式约束，得到一个通常的线性规划问题。由于 $\boldsymbol{x}^*$ 满足所有修改后的不等式，故 $\boldsymbol{x}^*$ 是该 LP 的

基可行解。它的检验数与原线性规划问题的数相同，说明该问题不能再改进。从而它是修改后的线性规划问题的最优解。故 $\boldsymbol{x}^*$ 在该转化成的LP问题的凸多面体的一个顶点上。又 $\boldsymbol{x}^*$ 满足所有的严格不等式，所以 $\boldsymbol{x}^*$ 不在这些严格不等式对应的超平面上。因此 $\boldsymbol{x}^*$ 在某一顶点上。

6.3 Linear-inequality feasibility

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

a. 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 grogramming problem should be polynomial in n and m.

b. 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.

解：(a) 只需要增加一个伪上标，应用线性规划算法即可。
(b) 只叙述方法。考虑线性规划问题（LP）和它对应的对偶问题（DLP）。对两上问题的不等式进行求解，并分别代入对应的目标。这样就得到最优目标所在的区间。取这个区间的中点作为最优解的一个估计，可增加一个不等式，构成新的不等式组。根据是否有解，确定新的最目标的区间范围。这样不断地迭代，直至达到要求的精度。

6.4 Linar Programming Modelling

INTEGER LINEAR PROGRAMMING PROBLEM is different from the classic Linear Programming Problem that some extra constraints such as
x_i is an integer, for all $i = 1, 2, \ldots, n$
are added.

A railway station has estimated that at least the following number of staff is needed in each four-hour interval throughout a standard 24-hour period and the salary per hour for every person during the different period:

Time Period	Staff Needed
5:00–9:00	S_1
9:00–13:00	S_2
13:00–17:00	S_3
17:00–21:00	S_4
21:00–1:00	S_5
1:00–5:00	S_6

Time Period	Salary Per Hour For Every Person
0:00–8:00	C_1
8:00–16:00	C_2
16:00–24:00	C_3

All staff works in 8-hour-shifts, which means every person will do continuous work for 8 hours. There are six possible shifts that start on the hour in the beginning of each 4-hour period in the table. Now if the railway station want to minimize the total salary paid in one day, please formulate this problem as an integer linear programming problem.

解：为方便描述，从5:00起每4个小时为一个时段，如段5:00–9:00称为第1时段；9:00–13:00称为第2时段。

设第i时段有y_i人工作，第i时段开始时有x_i人加入工作。因为每人一旦加入工作，就连续工作8小时，因此第i时段有x_{i-2}人离开工作（在模6同余的意义下计算脚标）。故按约束条件有

$$y_i = y_{i-1} - x_{i-2} + x_i \geq S_i,\ i = 1, 2, \ldots, 6.$$

下面分时间段考虑报酬：

- 0:00–8:00
  在这一时段，每人每工作一小时的报酬是C_1，首先在0:00–1:00这一个小时内，有y_5人工作；在1:00-5:00这四个小时内，有y_6人工作；在5:00-8:00这三个小时内有y_1人工作，因此，这一时间区间内的总报酬应为：

$$w_1 = C_1 y_5 + 4C_1 y_6 + 3C_1 y_1;$$

- 8:00–16:00
  同理分析，这一时间区间与第1、2、3时段相关，总报酬为：

$$w_2 = C_2 \cdot (y_1 + 4y_2 + 3y_3);$$

- 16:00–24:00
  同理，这一时间区间与第3、4、5时段相关，总的报酬为：

$$w_3 = C_3 \cdot (y_3 + 4y_4 + 3y_5).$$

因此全天的总报酬为$w = w_1 + w_2 + w_3$. 目标是使总的报酬最小，故最终问题形式为：

$$\begin{aligned}
\min\ w &= w_1 + w_2 + w_3, \\
y_i &= y_{i-1} - x_{i-2} + x_i \geq S_i,\ i = 1, 2, \ldots, 6, \\
w_2 &= C_1 \cdot (y_5 + 4y_6 + 3y_1), \\
w_2 &= C_2 \cdot (y_1 + 4y_2 + 3y_3), \\
w_3 &= C_3 \cdot (y_3 + 4y_4 + 3y_5), \\
x_i &\quad , \quad y_i\text{是非负整数}.i = 1, 2, \ldots, 6.
\end{aligned}$$

6.5 Linear Programming Modelling

0-1 Integer Programming Problem is different from the classic Linear Programming Problem that some extra constraints such as

$$x_i \in \{0, 1\}, \text{for all } i = 1, 2, \ldots, n$$

are added. There is an undirected graph G with n nodes. We use a matrix M to denote that whether two nodes are connected by an edge. In other words, m_{ij} is 1 if i and j are adjacent, and 0 otherwise. If you want to paint the nodes with some colors, such that any two adjacent nodes don't share the same color, and you try to use as less colors as possible, please formulate this problem as an integer linear programming problem (see the definition in Problem 4) or a 0-1 linear programming problem.

解：引入变量C_k

$$C_k = \begin{cases} 1, & \text{使用第}\, k\, \text{种颜色}, \\ 0, & \text{不使用第}\, k\, \text{种颜色}. \end{cases}$$

则0-1整数规划问题为：

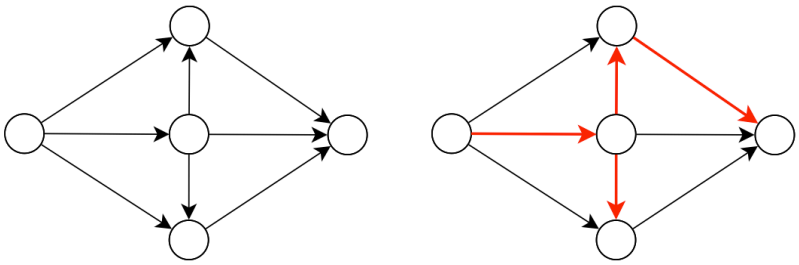
$$\min \sum_{i=1}^{n} C_k,$$

$$\text{s.t.} \begin{cases} m_{ij}(x_{ik} + x_{jk}) \leq m_{ij} C_k, \\ C_k \in \{0, 1\}, \\ x_{i,k} \in \{0, 1\},\ i, k = 1, 2, \ldots, n. \end{cases}$$

6.6 Linear Programming Modelling

In a directed acyclic graph, you can find no path which start from some node and stop at the same node. In this case, there are no Hamilton Cycle in this graph. In some practical problems, we will modify the conditions a little so as to find some approximate solutions.

For example, the original target can be modified to finding the minimum number of paths so that you're able to visit all nodes when you go through all these paths. The beginning and end of the paths are not restricted. In the following instance, the minimum number of paths is 2, and the paths are colored red.

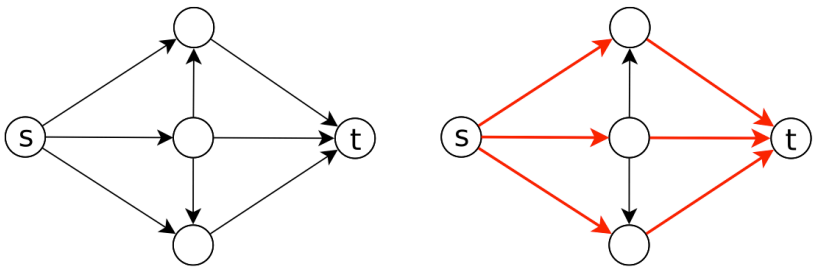


Try to formulate this modified problem as a linear programming problem or an integer linear programming problem (see the definition in Problem 4) or a 0-1 linear programming problem (see the definition in Problem 5).

解:此题比较困难,题意也不是很明确。我想可能与最小代价流(The min cost flow problem)有关。望已解决该问题的人给出解答[1]。

6.7 Linear Programming Modelling

FBI wants to send a very dangerous criminal from one prison to another. Just in case that the criminal be rescued, FBI decided to send several cars simultaneously to mislead his partners. The problem appears here: how to design the routes so as to arrange as many misleading cars as possible. Suppose all relative streets here are oneway streets. In this case, the problem can be transformed to finding as many paths as possible in a directed graph, while all these paths cannot share any node or edge except the starting point s and the finishing point t. In the following example, the maximum number of paths is 3, and the available paths are colored red.



Try to formulate this problem as a linear programming problem or an integer linear programming problem (see the definition in Problem 4) or a 0-1 linear programming problem (see the definition in Problem 5).

[1]可以联系我,我好将答案附上

解：归约为每个顶点（见问题7.1）、每条边容量为1的最大流问题。最终求得最大流数即最大路径条数。

7 网络流

7.1 Problem Reduction

In the Network Flow Problem, we add a group of constraints: for every node i, there has a capacity limitation k_i, the flow over the ith node cannot be greater than k_i. Please reduce the problem to the traditional Network Flow Problem.

解: 设该问题为 $P = (V, E, k, c, s, t)$. 其中 V 是有向图的顶点集，E 是边集，$k : V \to \mathbb{R}_+, k(i) = k_i, \forall i \in V;\ c : E \to \mathbb{R}_+$ 是容量。$s \in V$ 是源，$t \in V$ 是汇。

设 $V = \{s, t, v_1, v_2, \ldots, v_m\}, V' = \{s', t', v_1', v_2', \ldots, v_m'\}, \hat{V} = V \cup V'$. 令 $(v, v') \in E', \forall v \in V$. $\forall (u, v) \in E, (u', v) \in E'$

定义 $c' : E' \to \mathbb{R}_+$

$$c'(e) = \begin{cases} c((u, v)), & e = (u', v); \\ k(u), & e = (u, u'). \end{cases}$$

从而得到一最大流问题

$$\hat{P} = (\hat{V}, E', c', s, t)$$

对问题 $\hat{P}$ 的任意一个流，由 c' 的定义，流过任意顶点 $v \in V$ 的流量一定小于或等于 $k(v)$. 将 u, v' 收缩为一点，$\forall v \in V$, 就得到原问题 P 的一个流。因此只需求解传统的最大流问题 $\hat{P}$，然后转化为该问题的解即可。

7.2 Optimal cover problem

If each vertex of a graph G carries a positive weight, then the weight of a set of S of vertices is defined as the sum of weights of all the vertices in S and the *optimal cover problem* requires finding a cover of the smallest weight. Show that this problem can be turned into the minimum cut problem whenever G is bipartie.

解: 原问题的输入是一个二分图 G，输出是该二分图的一顶点子集 V_1, V_1 覆盖 G 所有的边，且 $w(V_1)$ 具有最小值。

我们将原问题转化为一最大流问题。令 $V(G) = X \cup Y$. 其中 X 中的任两点无边相连，Y 亦然。构造有向图 $G' : V(G') = V(G) \cup \{s, t\}$, 对任意无向边 $uv \in E(G)$, 作有向边 $(u, v) \in E(G')$; 添加胡向边 $(s, v), (u, t), \forall v \in X, u \in Y$ 至 $E(G')$ 定义 G' 上的容量函数：

$$c(u, v) = \begin{cases} +\infty, & uv \in E(G), \\ w(v), & u = s, \\ w(u), & v = t. \end{cases}$$

下面考虑 G' 的最小割具有怎样的形式。首先，如果 (S, T) 是网络流 G' 的最小割，那么 $\forall u \in S, v \in T$, 有 $(u, v) \notin E(G')$. 因为否则该割的容量 $c(S, T) = +\infty$. 但有一割

$(\{s\}, V(G') - s)$ 的容量为有限值。

下面我们将构造一个割，并且证明它是网络流 G' 的最小割。设 $A \in V(G)$ 是原问题的最优解，令

$$\begin{aligned} S &= (X \backslash A) \cup \{s\} \cup (Y \cap A), \\ T &= (X \cap A) \cup \{t\} \cup (Y \backslash A). \end{aligned}$$

显然有 $c(S,T) = w(A)$. 下面证明 (S,T) 是流 G' 的最小割。如若不然，存在另一割 (S',T') 且 $c(S',T') < c(S,T)$,

$$\begin{aligned} c(S',T') &= \sum_{v \in T'} c(s,v) + \sum_{u \in S'} c(u,t) \\ &= \sum_{v \in T' \cap X} c(s,v) + \sum_{u \in S' \cap Y} c(u,t) \end{aligned}$$

令 $B = (T' \cap X) \cup (S' \cap Y)$ 则

$$c(S',T') = w(B)$$

下面证明 B 是 G 的一个顶覆盖。$\forall e = uv \in E(G)$, 若 $u \in T' \cap X$, 则显然 e 已被覆盖。不妨设 $u \notin T' \cap X$ 且 $u \in X$. 则 $u \in X$
$(T' \cap X)$, 若 $v \notin S' \cap Y$, 那么存在有向边 $(u,v) \in E(G')$ 且 $u \in S', v \in T'$. 从而 $c(S',T') = +\infty$ 与 $c(S',T') < c(S,T)$ 矛盾。故 $v \in S' \cap Y$. 从而 $\forall e \in E(G)$. e 被 B 覆盖。又由 $c(S',T') = w(B) < c(S,T) = w(A)$, 与 A 是 G 的最优覆盖矛盾。从而 (S,T) 是网络流 G' 的最小割。

7.3 Problem Reduction

Let M be an $n \times n$ matrix with each entry equal to either 0 or 1. Let m_{ij} denote the entry in row i and column j. A diagonal entry is one of the form m_{ii} for some i.

Swapping rows i and j of the matrix M denotes the following action: we swap the values m_{ik} and m_{jk} for $k = 1, 2, \ldots, n$. Swapping two columns is defined analogously.

We say that M is rearrangeable if it is possible to swap some of the pairs of rows and some of the pairs of columns (in any sequence) so that, after all the swapping, all the diagonal entries of M are equal to 1.

(a) Give an example of a matrix M that is not rearrangeable, but for which at least one entry in each row and each column is equal to 1.

(b) Give a polynomial-time algorithm that determines whether a matrix M with 0-1 entries is rearrangeable.

解：(a) The matrix

$$\begin{pmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}$$

is not rerangeable.

(b) 令

$$\begin{aligned} V &= \{v_1, v_2, \ldots, v_n, v_{11}, v_{22}, \ldots, v_{nn}\} \cup \{s, t\} \\ E_0 &= \cup_{i=1}^{n} \{(s, v_i)\}, \\ E_1 &= \cup_{i=1}^{n} \{(v_i, v_{kk}) | m_{ik} = 1\}, \\ E_2 &= \cup_{i=1}^{n} \{(v_{ii}, t)\}, \\ E &= E_0 \cup E_1 \cup E_2, \\ c(e) &= \begin{cases} 1, & \forall e \in E_0 \cup E_2, \\ +\infty, & \forall e \in E_1 \end{cases} \end{aligned}$$

定义最大流问题 $P = (V, E, c, s, t)$. 如果问题 P 的最大流函数 f 满足 $f(s, V) = n$, 则矩阵 M 是可重排的，否则是不可重排的。因此只需要按照上面的方法，将问题转化成最大流问题，再用求解最大流问题的算法求解即可。

7.4 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 $s - t$ cut.

解： 应用 Ford-Fulkerson 算法计算最大流，并得到最小割 (S, T). 对每一条边 $e \in E(G)$ 且 $e = (u, v), u \in S, v \in T$. 将 (u, v) 的容量增加一个单位得新的网络流 G_e，如果对所有新的网络流的最大流有 $f_{G_e}(s, V) > f_G(s, V)$, 则唯一，否则不唯一。

7.5 Problem Reduction

A company has a collection of n software applications, $\{1, 2, \ldots, n\}$, running on its old system; and the president would like to port some of these to the new system. If he moves application i to the new system, they expect a net (monetary) benefit of $b_i \geq 0$. The different software applications interact with one another; if applications i and j have extensive interaction, then the company will incur an expense if they move one of i or j to the new system but not both; let's denote this expense by $x_{ij} \geq 0$.

Due to small but fundamental incompatibilities between the two systems, there's no way to port application 1 to the new system; it will have to remain on the old system. So,

this is the question: Which of the remaining applications, if any, should be moved? Give a polynomial-time algorithm to find a set $S \subset \{2, 3, \ldots, n\}$ for which the sum of the benefits minus the expense of moving the applications in S to the new system is maximized.

Hoping others to give a solution...

7.6 Maximum Cohesiveness

In sociology, one often studies a graph G in which nodes represent people and edges represent those who are friends with each other. Let's assume for purposes of this question that friendship is symmetric, so we can consider an undirected graph.

Now suppose we want to study this graph G, looking for a "close-knit" group of people. One way to formalize this notion would be as follows. For a subset S of nodes, let $e(S)$ denote the number of edges in S, that is, the number of edges that have both ends in S. We define the **cohesiveness** of S as $e(S)/|S|$. A natural thing to search for would be a set S of people achieving the maximum cohesiveness.

Give a polynomial-time algorithm that takes a rational number α and determines 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.

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.

8 近似算法

8.1 Approximation Algorithm

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$|A|/|C^*| \leq |C|/|C^*| \leq 2$.

8.2 Apptoximation Algorithm

Given a graph $G = V, E$ with edge costs and set $T \subseteq V$ of terminal 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).

(a)Give a 2-approximation algorithm if the edge costs satisfy the triangle inequality.

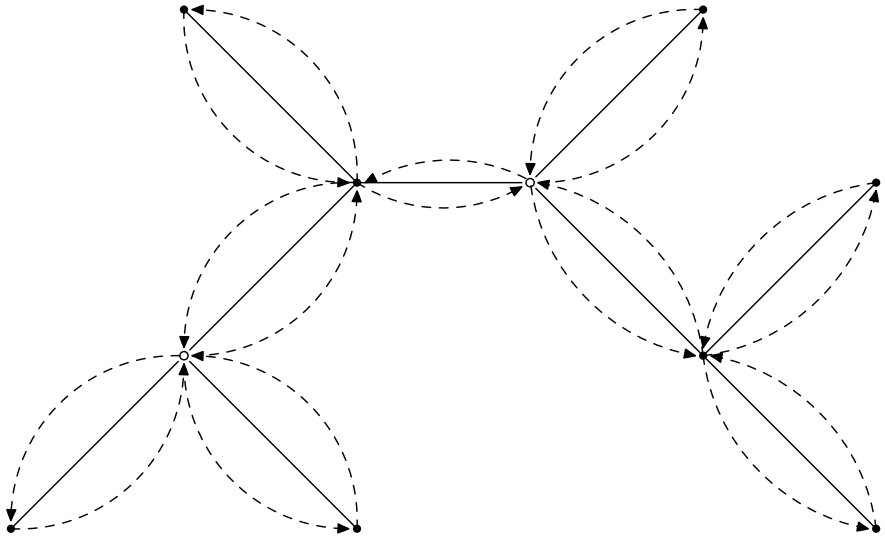
(b)Give a 2-approximation algorithm for general edge costs (The graph also need not be

complete).

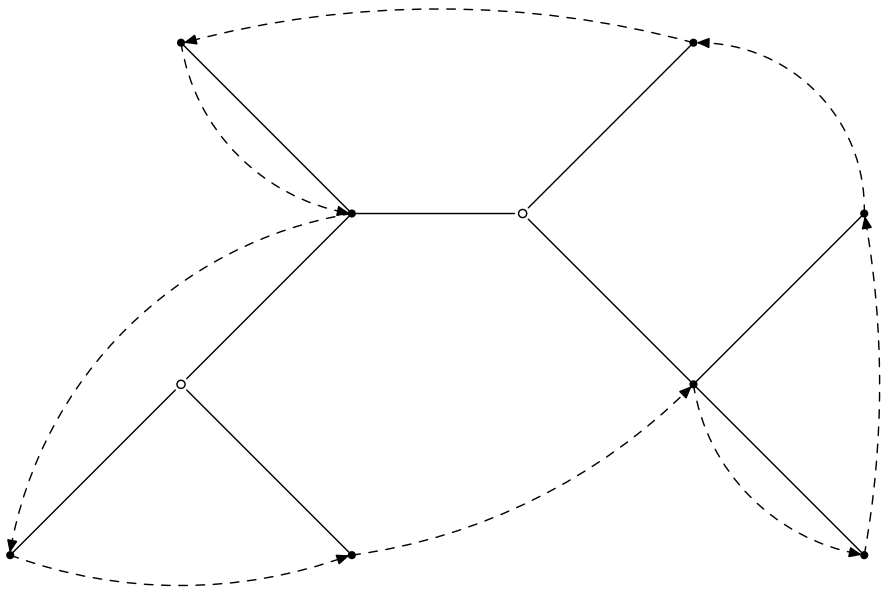
Solution: (a) We call the restricted (by the triangle inequality) problem *metric Steiner tree problem*

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 $2 \cdot \mathrm{OPT}$.

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 $2 \cdot \mathrm{OPT}$. 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 $2 \cdot \mathrm{OPT}$. This path is else a spanning tree on R. Hence, the MST on R has cost at most $2 \cdot \mathrm{OPT}$.

(b)Now I'll prove that *there is an approximation factor preserving reduction from the Steiner tree problem to the metric Steiner tree problem.*

We will transform, in polynomial time, an instance I of the Steiner tree problem, consisting of graph $G = (V, E)$, to an instance I' of the metric Steiner tree problem as follows. Let G' be the complete undirected graph on vertex set V. Define the cost of edge (u, v) in G' to be the cost of a shortest $u - v$ path in G. G' is called the metric closure of G. The partition of V into required and Steiner vertices in I' is the same as in I.

For any edge $(u, v) \in E$, its cost in G' is no more than its cost in G. Therefore, the cost of an optimal solution in I' dose not exceed the cost of an optimal solution in I.

Next, given a Steiner tree T' in I', we will show how to obtain, in polynomial time, a Steiner tree T in I of at most the same cost. The cost of an edge (u, v) in G' corresponds to the cost of a path in G. Replace each edge of T' by the corresponding path to obtain a subgraph of G. Clearly, in this subgraph, all the required vertices are connected. However, this subgraph may, in general, contain cycles. If so, remove edges to obtain tree T This completes the approximation factor preserving reduction.

8.3 Approximation Algorithm

Consider the following maximization version of the 3-Dimensional Matching Problem. Given disjoint sets X, Y, Z, and given a set $T \subseteq X \times Y \times Z$ of ordered triples, a subset $M \subseteq T$ is a $3 - dimensional\ matching$ if each element of $X \cup Y \cup Z$ is contained in at most one of these triples. The $Maximum\ 3 - Dimensional\ Matching\ Problem$ is to find a 3-dimensional matching M of maximum size. (You may assume $|X| = |Y| = |Z|$ if you want.)

Give a polynomial-time algorithm that finds a 3-dimensional matching of size at least $\frac{1}{3}$ times the maximum possible size.

Solution: The trivial algorithm below approximates the optimal solution within $1/3$. (For any triple in the found matching, consider it and all its neighbor triples. At most three of these triples can be in the optimal matching.)

$T = M; M = \emptyset;$
while $T \neq \emptyset$ **do**
  $t =$ any element in T ;

$M = M \cup \{t\}$;
$T = T\{t$and all its neighbors in $T\}$;
end while;

8.4 Approximation Algorithm

Consider an optimization version of the Hitting Set Problem defined as follows. We are given a set $A = \{a_1,\ a_2, ..., a_n\}$ and a collection $B_1,\ B_2, ..., B_m$ or subsets of A. Also, each element $a_i \in A$ has a $weight\ w_i \geq 0$. The problem is to find a hitting set $H \subseteq A$ such that the total weight of the elements in H, that is, $\Sigma_{a_i \in H} w_i$, is as small as possible. (H is a hitting set if $H \cap B_i$ is not empty for each i.) Let $b = max_i|B_i|$ denote the maximum size of any of the sets $B_1, B_2, \ldots, B_m$. Give a polynomial-time approximation algorithm for this problem that finds a hitting set whose total weight is at most b times the minimum possible.

Solution:

The hitting set problem is equivalent to the set cover problem: An instance of set cover can be viewed as an arbitrary bipartite graph, with sets represented by vertices on the left, elements of the universe represented by vertices on the right, and edges representing the inclusion of elements in sets. The task is then to find a minimum cardinality subset of left-vertices which covers all of the right-vertices. In the hitting set problem, the objective is to cover the left-vertices using a minimum subset of the right vertices. Converting from one problem to the other is therefore achieved by interchanging the two sets of vertices.

Since we have given a 2-approximate algorithm to solve the set cover problem (see *Algorithms, T.Cormen et al.*), we get a 2-approximate algorithm for the hitting set problem.

8.5 Approximation Algorithm

Suppose you are given a set of positive integers $A = \{a_1, a_2, \ldots, a_n\}$ and a positive integer B. A subset $S \subseteq A$ is called feasible if the sum of the numbers in S does not exceed B:

$$\sum_{a_i \in S} a_i$$

You would like to select a feasible subset S of A whose total sum is as large as possible.

Example: If $A = \{8, 2, 4\}$ and $B = 11$, then the optimal solution is the subset $S = \{8, 2\}$

Give a polynomial-time approximation algorithm for this problem with the following guarantee: It returns a feasible set $S \subseteq A$ whose total sum is at least half as large as the maximum total sum of any feasible set $S \subseteq A$. Your algorithm should have a running tim of most $O(n \log n)$.

解：方法如下：从大到小排序，如果可能，依次将元素加入集合。如果$\exists x \in A$使得$B \geq x \geq B/2$,则结论显然成立。否则，设排好序后的次序为$B/2 \geq x_1, x_2, \ldots, x_n$,必可从中选出一组其和大于或等于$B/2$或者全部被选取（这时达到最优）。

8.6 Approximation Algorithm

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

(a) Give a 2-approximation algorithm for this problem.

(b) Prove the following theorem:
For any $\varepsilon > 0$, there is no approximation algorithm having a guarantee of $\frac{3}{2} - \varepsilon$ for the bin packing problem, assuming $\mathbf{P} \neq \mathbf{NP}$.

You can use any conclusion in *Assignment 2*.

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
  $\sum_{i=1}^{k} a_i = \sum_{i=1}^{k-1} v_i + v_k > \sum_{i=1}^{k-1}(1 - v_k) + v_k > (k-1) - (k-2)v_k > k/2.$

(b) *Hoping others to give a solution....*