ZOJ Monthly, December 2008解题报告

watashi@zju*

December 31, 2008

Contents

*Zejun.WU@gmail.com

ZOJ Monthly, December 2008

Description: ZOJ Monthly, December 2008
Start Time: 2008-12-27 12:00:00 (GMT+8)
Length: 5 Hours

ID	Title	Ratio (AC/All)
A	Bernstein Polynomial	0.00% (0/7)
B	Pyraminx	37.80% (31/82)
C	Ticket	19.25% (31/161)
D	Team Forming	6.45% (2/31)
E	Move to Baggage Office	8.92% (19/213)
F	Greedy	10.71% (6/56)
G	Text	13.48% (12/89)
H	ChiBi	11.41% (54/473)
I	The Grid	0.00% (0/17)

声明：如果这个解题报告对您有帮助，它是watashi写的，否则我也知道是谁写的

错别字请54

A Bernstein Polynomial

3073 Bernstein Polynomial

Time Limit: 10 Seconds Memory Limit: 32768 KB

Author: **WU, Zejun**

题目要求一个多项式$p(x)$的n次Bernstein多项式

$$B_n(p) = \sum_{i=0}^{n} p(\frac{i}{n}) \binom{n}{i} x^i (1-x)^{n-i} \tag{1}$$

并要求保留高精度分数形式。其中，多项式的度$d < 64$，而$n < 65536$。

直接根据定义求的复杂度是$O(n^2)$的，不满足要求。实际应该选用复杂度为$O(d^2)$的算法。在介绍算法以前先证明Bernstein多项式成立以下定理。

Theorem A.1.

$$B_n(x^k) = \sum_{j=0}^{k} \frac{(n)_j}{n^k} S(k,j) x^j \tag{2}$$

其中$(n)_j$是*falling factorial*，$S(k,j)$是*Stirling numbers of the second kind*。

Proof. 将恒等式$i^k = \sum_{j=0}^{k} S(k,j)(i)_j$（证明略）带入定义中

$$\begin{aligned}
B_n(x^k) &= \frac{1}{n^k} \sum_{i=0}^{n} \sum_{j=0}^{k} S(k,j)(i)_j \binom{n}{i} x^i (1-x)^{n-i} \\
&= \frac{1}{n^k} \sum_{j=0}^{k} S(k,j) \sum_{i=0}^{n} (i)_j \binom{n}{i} x^i (1-x)^{n-i} \\
&= \frac{1}{n^k} \sum_{j=0}^{k} S(k,j) \sum_{i=j}^{n} \frac{i!}{(i-j)!} \frac{n!}{i!(n-i)!} x^i (1-x)^{n-i} \\
&= \frac{1}{n^k} \sum_{j=0}^{k} S(k,j) x^j \sum_{i=0}^{n-j} \frac{n!}{(n-j)!} \binom{n-j}{i} x^i (1-x)^{n-j-i} \\
&= \frac{1}{n^k} \sum_{j=0}^{k} S(k,j) x^j (n)_j
\end{aligned}$$

□

对于Stirling numbers of the second kind有

$$S(k,j) = S(k-1,j-1) + jS(k-1,j), with S(k,1) = S(k,k) = 1.$$

可以$O(d^2)$预处理成一张表。用上面的公式求$B_n(x^k)$的复杂度为$O(k)$，对多项式的每一项通过上述公式求解，再利用$B_n(P) + B_n(Q) = B_n(P+Q)$（证明显然）通过多项式的数乘和加法运算，我们得到了$O(d^2)$的算法。

当然，通过上述证明我们可以知道对于次数大于d的项，Bernstein多项式的系数始终为0。因此忽略这些项系数的计算，直接用定义求，复杂度也是$O(d^2)$的。此时的公式可以写成

$$B_n(p) = \sum_{i=0}^{d} p(\frac{i}{n}) \binom{n}{i} x^i \sum_{j=0}^{d-i} (-1)^j \binom{n-i}{j} x^j, \quad d \leq n$$

当然，这时候不可能再把所有的$\binom{n}{i}$预处理出来，而要通过双重循环过程中的技巧来做到$O(1)$得到需要的$\binom{n}{i}$。当然，如果能观察得到或猜测没有次数大于d的项，也能得到以上算法。

由于题目要求高精度分数，建议用Java实现。

B Pyraminx

3074 Pyraminx

Time Limit: 1 Second Memory Limit: 32768 KB

Author: **GAO, Yuan**

已知正四面体的初状态和一些操作规则，给定一系列操作，要求打印末状态。

只要把变换规则转成片与片位置变换的数组或代码，并且注意输入输出格式就可以了。比较人肉的一道题，至于如何把代码写的短，写得清晰就仁者见仁，智者见智了。

C Ticket

3075 Ticket

Time Limit: 3 Seconds Memory Limit: 32768 KB

Author: **LI, Cheng**

假设校icpc队员要训练iP至Q（包含）天，每人每天的午餐费用为F，已知各个时间的机票价，问最优方案下安排队员乘机来回和**训练外**午餐的最小花费。

数据规模要求要用一个$O(n \lg n)$或$O(n)$的算法。

C.1

首先做如下处理，对于来学校的机票，假如一天的机票几个比前面某天的机票价格加这几天的饭钱要贵，那么可以用前面某天的机票价格加这几天的饭钱代替这一天的机票价格。对于回学校的机票则正好相反。经过这些$O(n)$的预处理后，我们不再需要考虑吃饭问题，于是问题就是找出长度为P至Q的子序列，使得两端的和最小。解决问题可以先枚举左端，然后就知道右端合法位置的区间，通过区间段最小值查询(RMQ)，我们可以快速地得到该区间内的最小值。这个算法的复杂度是$O(n \lg n)$。注意不要MLE。

TICKETS-RMQ($tickets, P, Q, F$)

```
1  n ← |tickets|
2  come ← tickets
3  for i ← 2 to n
4        do come[i] ← min {come[i], come[i − 1] + F}
5  go ← tickets
6  for i ← n − 1 downto 1
7        do go[i] ← min {go[i], go[i + 1] + F}
8  RMQ-INITIALIZE(go)
9  ans ← ∞
10 for i ← 1 to n − 1 − P
11       do ans ← min{ans,
             come[i] + RMQ-QUERY(i + P + 1, min {i + Q + 2, n + 1})
12 return ans
```

C.2

维护两个OrderedMultiSet(如用std::multiset)，其中$st1$保存$P+1$至$Q+1$天前的所有票价$tickets[i]$，$st2$保存所有多于$Q+2$天前的票价减去$i-1$天的饭钱的值，原因参考后面的分析。然后扫描$tickets$，每次更新$st1$和$st2$，

假设扫描到第k天，就用$tickets[k] + \min\{st1\}$和$tickets[k] + (k - Q) * F + \min\{st2\}$更新答案。这里就可以看出，插入$st2$元素是票价减去$i - 1$天的饭钱的值的处理是为了能够方便地对所有$k$和$i$计算饭钱，其实这和上面算法的预处理效果是一样的。对$(k - Q) * F - (i - 1) * F$也可以理解为$k$天里$Q$天吃白饭$i - 1$天不吃饭的花费。复杂度$O(n \lg n)$。

C.3

算法同上，但是注意到其实我们可以不需要维护OrderedMultiSet。对于$st2$，只需要记录一个最小值就完全足够了。对于$st1$，可以维护一个关于i单调递增，关于$ticket[i]$ 单调递减的双端队列。于是复杂度就是$O(n)$的了。

D Team Forming

3076 Team Forming

Time Limit: 5 Seconds Memory Limit: 32768 KB

Author: **XIAO, Dong**

给$N \leq 18$个人在$A \leq 20$个方面的能力值，假设每个队伍的每项能力值都是三个队员该项能力的最高值，把这A项能力按顺序画出的雷达图的面积作为队伍的总能力值。求使得前$F = 1$ *or* 2队总能力值最高的前提下所有队伍总能力和最高的分组方案。

关于精度问题，这题并没有在这方面设卡，不过仍值得一提。首先是输入数据全部可以乘100转化为整数。而对于雷达图的面积，由于只用作比较，可以直接用公式

$$a_1 a_n + \sum_{i=1}^{n-1} a_i a_{i+1}$$

求得，也就是不需要乘以常系数$\frac{1}{2}sin(\frac{2\pi}{N})$。

因为N最大18，能很直观的想到2^{18}的DP/搜索。首先可以通过暴力算法预处理出所有$\binom{n}{3}$种队伍组合的能力值。然后通过集合上的DP得出能力和的最大值。对于前$F = 1$ *or* 2队总能力值最高的前提，可以通过把非最大的DP值都置为-1之类的无效值来防止其状态继续转移。理论复杂度**ms**是$O(2^N N^3)$，但是其常数很小，而且前F队能力和最大的限制可以让有效状态大大减小，还可以通过预处理出$\binom{N}{3k}$的mask 值来进行加速。

引用作者的话

> 标程跑120多组随机case只用了390ms，跑2组所有能力都一样的case也才400ms左右，2^{18}的方法又预处理了$\binom{18}{3}$的面积的话比赛时应该都能在2s内跑出。

E Move to Baggage Office

3077 Move to Baggage Office
Time Limit: 1 Second Memory Limit: 32768 KB
Author: **LI, Yaochun**

帮运价值v_i为i包至少需要a_i的体力，会用去$a_i - b_i$的体力，已知初始体力s，求最大可能获得的价值。

比赛时的一个坑，只有不到10%的AC率。

这不是一个单纯的背包问题，不能直接dp，因为这样不满足无后效性。不过这种问题可以通过按一定的顺序排序后再dp来解决。

按$a-b$升序排列是不对的，也是WA得最多的。反例：

```
1
8 3
1 3 2
2 5 3
4 8 5
```
正确答案是7
错误输出为4

按a降序排也是错的。反例：

```
1
22 2
1 21 18
2 20 19
```
正确答案为3
错误输出为2

正错的策略是**按b降序排**。当然没有反例……

> 顺带一提如果不确定如何排序，可以ws地用用n种排序各求一个解，再返回其中最大的。然后就应该能AC了。-,-

错误方法就不讨论了，下面证明按b降序排可以得到正确结果。显然按b降序排得到的结果本身就是一个可能的解，所以不大于最优解；另一方面，任意一种合法的方案，都对应有一个集合相同，顺序按b降序的方案（待证），即求得的结果不小于最优解。所以求得结果就是最优解。对于第二部分的证明，不失一般性，我们假设一个最优方案就是依次搬运1至n号包。

Theorem E.1. 若排列$1, 2, \cdots, n$满足

$$s - \sum_{i=1}^{k-1} (a_i - b_i) \geq a_k, \quad k = 1, 2, \cdot, n a_i \geq b_i, \quad k = 1, 2, \cdot, n$$

那么按b降序排后得到的排列$p_1, p_2, \cdots, p_n$也满足该性质，即有

$$s - \sum_{i=1}^{k-1} (a_{p_i} - b_{p_i}) \geq a_{p_k}, \quad k = 1, 2, \cdot, n$$

Proof. 若排列$1, 2, \cdots, n$已经是按b降序排列的，结论显然成立。

否则，$\exists j$ s.t. $b_j < b_{j+1}$，考虑交换j和$j+1$的新排列。同原来的排列比较可知在$k \neq j, k \neq j+1$时性质显然成立。记$s' \triangleq s - \sum_{i=1}^{j-1} (a_i - b_i)$，我们只需要证明

$$\left\{ \begin{array}{rcl} s' & \geq & a_j \\ s' - (a_j - b_j) & \geq & a_{j+1} \end{array} \right. \Rightarrow \left\{ \begin{array}{rcll} s' & \geq & a_{j+1} & (1) \\ s' - (a_{j+1} - b_{j+1}) & \geq & a_j & (2) \end{array} \right.$$

由$s' \geq (a_j - b_j) + a_{j+1} \geq 0 + a_{j+1} \geq a_{j+1}$，(1)式得证。根据上述假设$b_j < b_{j+1}$，所以

$$\begin{array}{rcl} & s' - (a_j - b_j) & \geq & a_{j+1} \\ \Leftrightarrow & s' - a_{j+1} + b_j & \geq & a_j \\ \Rightarrow & s' - (a_{j+1} - b_{j+1}) & > & a_j \end{array}$$

(2)式得证。所以交换j和$j+1$的新排列也满足定理中的性质。

只要序列不是按b降序排列的我们就可以不断交换排列中的两个数，而保持定理中的性质不变。最后，这就像冒泡排序一样，我们可以把序列按b降序排列而保持性质不变。也就是说对任意一个可行的方案，在按b降序排列后，依然是可行的。

□

F Greedy

3078 Greedy

Time Limit: 1 Second Memory Limit: 32768 KB

Author: **LIN, Yue**

给定一个有向图，watashi的mm(求mm……)至多可以$0 \leq Q \leq 10$次从1号城市出发经过标号在1到$n-1$之间的城市后走到n号城市。其收益是曾路过的各个城市的加权和，要求最大收益。

拆点费用流模型。对于标号在1到$n-1$之间的点i都拆成两个点ia和ib，其中ia没有到ib以外的出边，ib没有来自ia以外的入边。ia到ib引两条边，一条容量为1 费用为$-w[i]$（权的相反数），另一条容量为∞费用为0。对于原图中的i到j的有向边，在新图中添加一条ib到ja容量为∞费用为0的边。增加一个源点，向1引一条容量为Q费用为0的边；n就是汇点，没有出边，如果原来有就删掉。然后求一次最小费用最大流，答案就是求得的最小费用的相反数。

G Text

3079 Text

Time Limit: 1 Second Memory Limit: 32768 KB

Author: **MO, Luyi**

字符串处理题，按照通常编辑器里对英语的处理标准，将一段格式混乱的段落格式化。一个例外是'-'结尾的字符串要和下一个字符串合并。所有的输入都是合法的。

字符串处理离不开对各种情况的充分考虑和细心的coding，也是仁者见仁，智者见智的。这题的数据规模很小，基本不用担心TL和ML，还是健壮性第一。

这里提供一个程序，供对拍用。

```
#include <cctype>
#include <cstdio>
#include <cstring>

bool begin;
int length;

bool gao()
{
        static int len;
        static char ch, buf[1024], buff[1024];

        scanf("␣");
        if (scanf("%[#]", buff) == 1) {
                return false;
        }
        if (scanf("%[A-Za-z]", buf) != 1) {
                throw -1;
        }
        ch = getchar();
        if (ch == '-') {
                scanf("␣%[A-Za-z]", buff);
                strcat(buf, buff);
                ch = '␣';
        }
        if (begin) {
                buf[0] = toupper(buf[0]);
                begin = false;
        }
        if (isspace(ch) && scanf("␣%1[,.?!]", buff) == 1) {
                ch = buff[0];
        }
        if (!isspace(ch)) {
                if (ch != ',') {
```

```
                        begin = true;
                }
                buff[0] = ch;
                buff[1] = '\0';
                strcat(buf, buff);
        }
        len = strlen(buf);
        if (length + len + (length != 0) <= 80) {
                if (length != 0) {
                        putchar(' ');
                }
                printf("%s", buf);
                length = length + len + (length != 0);
        }
        else {
                printf("\n%s", buf);
                length = len;
        }

        return true;
}

int main(void) {
        int re;

        scanf("%d", &re);
        while (re--) {
                begin = true;
                length = 0;
                while (gao());
                putchar('\n');
                if (re > 0) {
                        putchar('\n');
                }
        }

        return 0;
}
```

H ChiBi

3080 ChiBi

Time Limit: 5 Seconds Memory Limit: 32768 KB

Author: **CHAO, Jiansong**

给定一些船，如果一艘船起火，则所有与之相连的船都会起火。现在知道士兵去点燃各船的距离和船与船之间传播火需要的时间，要求在保证用最少人次士兵的前提下，使的烧光所有船的时间最少，输出该时间。

题目中很重要的条件是连通分量的大小不超过100，保证用最少人次士兵的前提就是说每个连同分量只能点燃一次，于是问题就很简单了。首先通过dfs或bfs找出连通分量，再对连通分量内所有点求一次floyd，这样可以得到一个连通分量及其两点之间的最短路径。接着枚举点燃连通分量里的哪艘船，求得这样烧掉整个连通分量所需的时间，用所有枚举求得的最小值去更新答案。

其他算法就不介绍了。

I The Grid

3081 The Grid

Time Limit: 1 Second Memory Limit: 32768 KB

Author: **FAN, Yuzhe**

8数码问题的三维加强版

以下直接引用作者的解题报告:

本题标准程序使用了基于迭代加深的启发式搜索算法(IDA*)，实际上是经典的8数码问题的加强版，将原来的二维平面问题扩展到了三维。对于经典的8数码问题，目前已知最成熟又最容易实现的方法就是IDA*算法。对于非empty的每一个数码A，设其初始位置是(x, y, z)，目标位置是(x', y', z')，则可以定义每个A的完成距离为

$$dist(A) = |x - x'| + |y - y'| + |z - z'|$$

则可以定义一个估价函数

$$TotalDist(CurrentState) = \sum_{i=1}^{P*Q*R} dist(A_i)$$

由于在这个函数中没有考虑空格位置的完成距离，所以每一次移动操作将使这个估价函数变动至多1，即$TotalDist(CurrentState)$是$CurrentState$的一个很接近精确的下界，因此对于绝大多数有解的问题都可以较快的找到最优解(关于IDA*算法取得最优解的性质请大家自行思考或查询资料)。

以上算法能够很好的解决有解的问题，对于无解的问题，可以利用这个问题的性质来直接证明无解。

将所有数按照从前到后从上到下从左到右的顺序（即题目Input输入数据的顺序）排列，定义为S。则最终状态为$S = (1, 2, 3, ..., P * Q * R)$。对于每一种移动方法考虑排列$S$的变化（设空格处于位置$i$）：

- 空格前移，相当于元素i和元素$i - Q * R$交换
- 空格后移，相当于元素i和元素$i + Q * R$交换
- 空格上移，相当于元素i和元素$i - R$交换
- 空格下移，相当于元素i和元素$i + R$交换
- 空格左移，相当于元素i和元素$i - 1$交换
- 空格右移，相当于元素i和元素$i + 1$交换

考虑S经过一次操作后的逆序对数的变化。以元素i(已标记为最大值的空格元素)和元素$i-R$交换为例，假设在$(i-R,i)$区间内有a个数小于元素$i-R$，则S的逆序对数的变化为$(((R-1)-a)+R)-a=(R-a)*2-1$为奇数，同时$dist(P*Q*R)$的变化绝对值为1，从而有任意的操作不改变$S$的逆序对数+$dist(P*Q*R)$的总和的奇偶性这一性质。易于知道本题描述中最终状态的此值是偶数。利用这一性质，我们可以在$O((P*Q*R)^2)$的时间内判断一个初始状态是否能够得到最终态。至此，这个问题得到了比较完整的解决。

IDA*算法的简要介绍: IDA*(Iterative Deepening A*)指的是一种基于迭代的通过不断加深搜索深度，以利用搜索深度进行强剪枝的搜索算法。这一算法能够利用枚举加深的深度值，来剪去状态空间中估计会超出最大可能深度的状态节点。对比其它朴素搜索方式，这种搜索具有不需要判重，不需要排序(对于启发式搜索)，空间需求与搜索树大小具有对数关系等优点。详细算法结构可以参考刘汝佳黑书的P172和P190，对于这一算法有较具体的描述。