算法分析习题选讲(第三章)

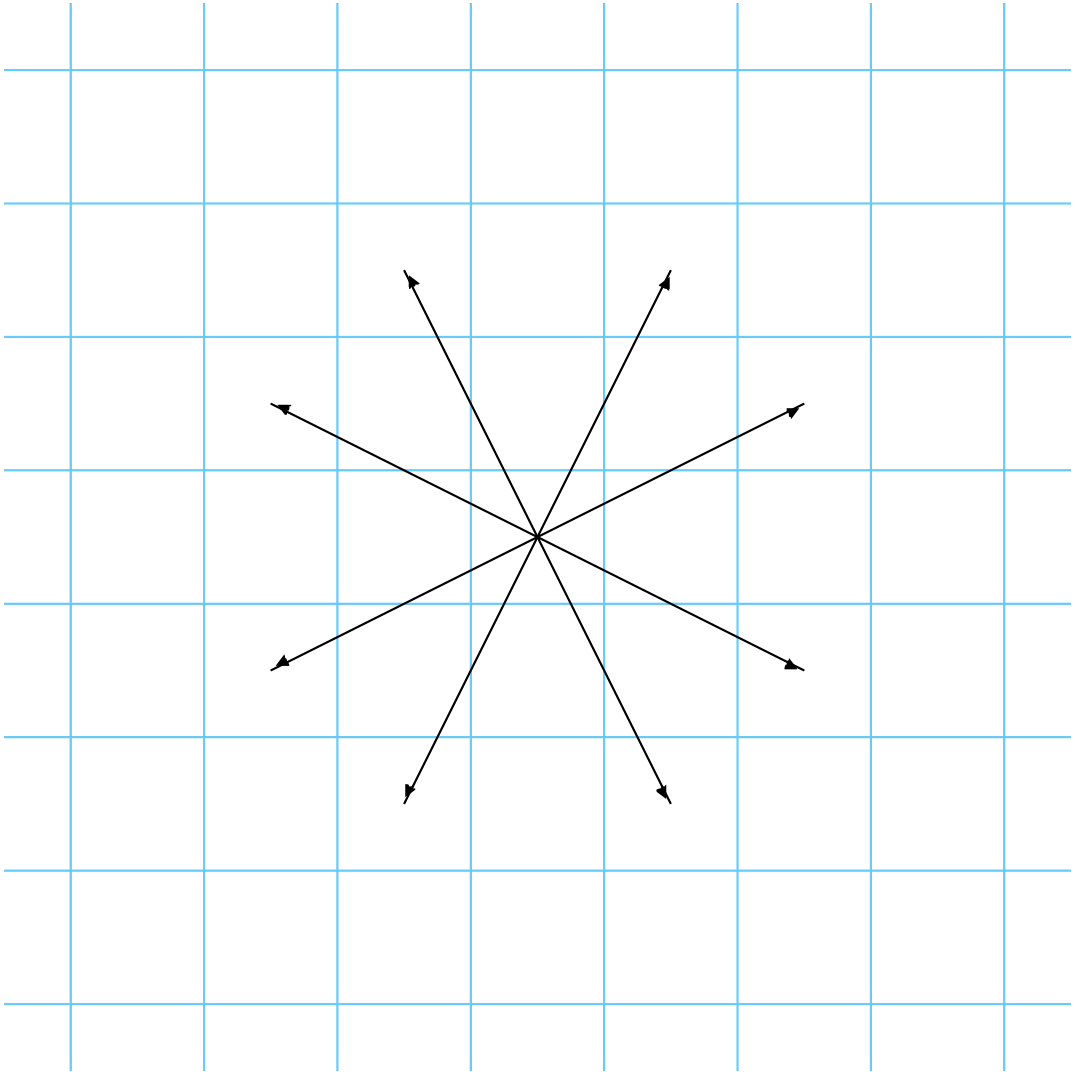
chyx111@qq.com

1152 1153 马周游

1152 1153 马周游 题目大意

一个有限大小的棋盘上有一只马

给出初始时马的位置，找出一条马移动的路线，经过所有格子各一次



1152 1153 马周游 题目大意

1	2	3	**4**	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64

1152 1153 马周游 题目大意

1	2	3	**4**	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	**21**	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64

1152 1153 马周游 题目大意

1	2	3	**4**	5	**6**	7	8
9	10	11	12	13	14	15	16
17	18	19	20	**21**	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64

1152 1153 马周游 解题思路

- 深搜
- 枚举马能走的所有路径，直至找到一条完成周游的路径
- 回溯

1152 1153 马周游 代码

```
bool Solve(int x, int depth) {
  route[depth] = x + 1;
  if (depth == m * n - 1) {
    print_route();
    return true;
  }
  visit[x] = true;

  //搜索对效率要求较高，建议将这里换成int children[8]以提高效率。
  vector<int> children;

  get_children(x, &children);
  for (int i = 0; i < children.size(); ++i) {
    if (Solve(children[i], depth + 1)) return true;
  }

  visit[x] = false;
  return false;
}

void get_children(int x, vector<int> *children) {
  for (int i = 0; i < neighbors[x].size(); ++i) {
    int child = neighbors[x][i];
    if (!visit[child]) {
      children->push_back(child);
    }
  }
}
```

1152 1153 马周游 缺点

程序过慢，只能勉强过1152

优化：改变搜索顺序

先搜索可行格较少的格子

x . x x x

x x x . .

. x ?(2) x x

x x x . .

x . ?(1) x x

1152 1153 马周游 代码

```
int cnt_size[64];

bool cmp(int x, int y) {
  return cnt_size[x] < cnt_size[y];
}

int get_children_size(int x) {
  int size = 0;
  for (int i = 0; i < neighbors[x].size(); ++i) {
    int child = neighbors[x][i];
    if (!visit[child]) {
      ++size;
    }
  }
  return size;
}

void get_children(int x, vector<int> *children) {
  for (int i = 0; i < neighbors[x].size(); ++i) {
    int child = neighbors[x][i];
    if (!visit[child]) {
      children->push_back(child);
      cnt_size[child] = get_children_size(child);
    }
  }
  sort(children->begin(), children->end(), cmp);
}
```

1152 1153 马周游 解题报告

- 可在解题报告中尝试其他搜索顺序或剪枝，对比其效果
- 通过加大数据范围，如扩展到9x9, 10x10，本地跑程序来对比不同算法的性能
- 可以思考构造性的算法

1093 Air Express

1093 Air Express 题目大意

给出4个重量区间 & 每个区间的单位重量运输价格

Package weight	Cost per pound
0 to 9 pounds	$10
10 to 49 pounds	$5
50 to 99 pounds	$3
100 pounds or more	$2

有一个背包需要运输，问往背包里面添加多少重量后可以让运费最低

1093 Air Express 解题思路

Package weight	Cost per pound
0 to 9 pounds	$10
10 to 49 pounds	$5
50 to 99 pounds	$3
100 pounds or more	$2

最小运输价格必定出现在：

1. 不添加任何重量
2. 添加重量后刚好到达某个区间的下界

1093 Air Express 代码

```
int cal(int weight) {
  int price = INF;
  for (int i = 0; i < 4; i++) {
    if (lower[i] <= weight && weight <= upper[i]) {
      price = min(price, weigth * rate[i]);
    } else if (weight < lower[i]) {
      price = min(price, lower[i] * rate[i]);
    }
  }
  return price;
}
```

修改这段代码让它输出需要添加的重量

1134 积木分发

1134 积木分发 题目大意

n个小伙伴，每个人手上有a_i块积木，还需要b_i块积木才能完成任务

The Pancakes手上有s块积木，她可以把她手中的积木都给某个人，等那个人完成任务后回收他手上的所有积木

问The Pancakes最后是否能回收完所有人的积木

$s \leq 10^6, n \leq 10^4, a, b \leq 10^9$

1134 积木分发 样例

第一个样例：

n = 2, s = 2

a = 1, b = 4

a = 2, b = 1

分给第二个人 -> s = 4 -> 再分给第一个人 -> s = 5

第二个样例：

n = 2, s = 2

a = 1, b = 4

a = 1, b = 1

分给第二个人 -> s = 3 -> 第一人仍然不够，失败

1134 积木分发 解题思路

应该先分给需求少的人，因为分完后The Pancakes手上的积木总是会变多的

排序后贪心求解

1134 积木分发 代码

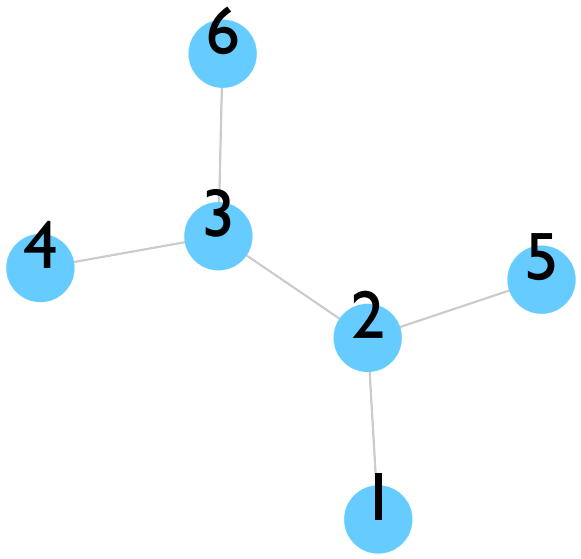
```
struct Node {
  int have, need;
};
bool operator<(const Node& x, const Node& y) {
  return x.need < y.need;
}
bool Solve() {
  sort(nodes, nodes + n);
  for (int i = 0; i < n; i++) {
    if (s < nodes[i].need) return false;
    s += nodes[i].have;
  }
  return true;
}
```

1140 国王的遗产

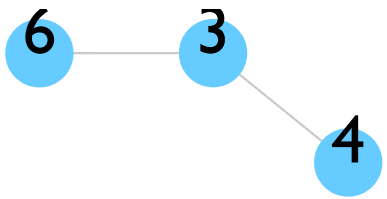
1140 国王的遗产 题目大意

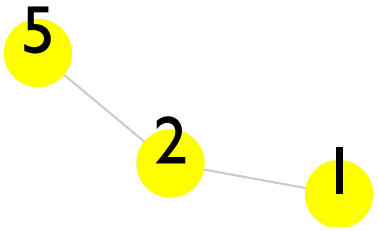
一棵由n块金块组成的树

k个人按顺序轮流拿金块，每个人拿的时候选择树的一条边将其分割成两棵树



1140 国王的遗产 题目大意



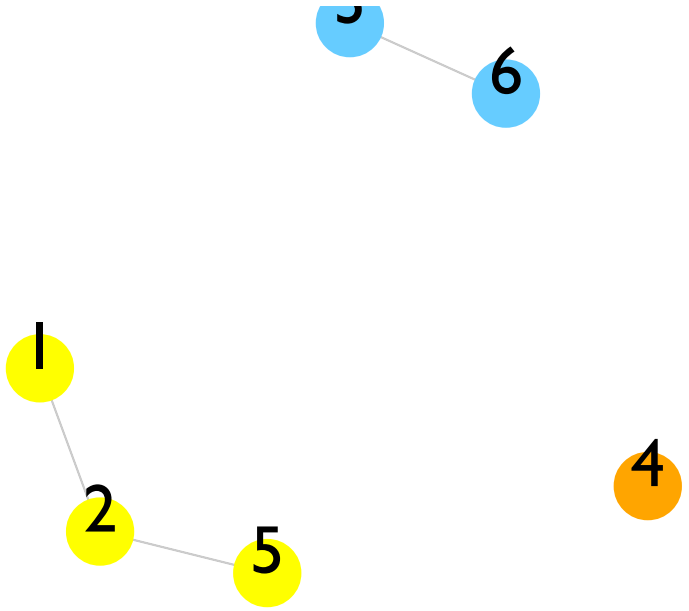


拿的那部分的金块数不能超过当前金块总数的一半

每个人都希望拿到尽量多的金块

如果有多种拿法，则拿最小金块编号最小的那一块

1140 国王的遗产 题目大意



1140 国王的遗产 题目大意

```
6 3
1 2
2 3
3 4
2 5
3 6

3 1 2
```

$n \leq 30000, k \leq 100$

1140 国王的遗产 解题思路

按顺序做，枚举每一个人，检查切断每一条边所得到的两棵子树，计算其节点数和最小编号

如何得到这两棵子树？

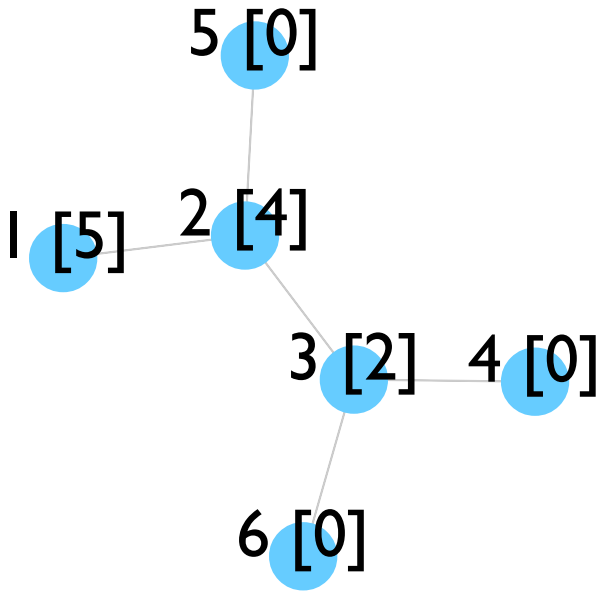
去掉边后做DFS

复杂度？

$O(N^2K)$, 枚举边 * DFS求大小 * K次，无法承受

1140 国王的遗产 解题思路

转化成有根树，只需要做一次DFS



1140 国王的遗产 代码

记录子树的大小，删除的边的两个端点，子树中的最小编号

```
struct SubTree {
  int num_nodes;
  int insider, outsider;
  int min_id;
};
```

用vector来保存树的边

```
const int kMaxN = 31000;
vector<int> G[kMaxN];
SubTree subtree[kMaxN];
```

1140 国王的遗产 代码

不同子树之间的比较：

```
bool operator<(const SubTree& a, const SubTree& b) {
  if (a.num_nodes != b.num_nodes) {
    return a.num_nodes > b.num_nodes;
  }
  return a.min_id < b.min_id;
}
```

主过程：

```
vector<int> ans;
pivot = 0;
total = n;
for (int ichild = 0; ichild < num_children - 1; ++ichild) {
  pivot = dfs_find_min(pivot, -1);

  best.num_nodes = -1;
  dfs(pivot, -1);
  ans.push_back(best.num_nodes);

  // Remove edge best.insider <-> best.outsider

  total -= best.num_nodes;
  pivot = best.outsider;
}
```

1140 国王的遗产 代码

找出树中的最小编号，从该编号开始DFS

```
int dfs_find_min(int x, int parent) {
  int min_id = x;
  for (int i = 0; i < G[x].size(); ++i) {
    int child = G[x][i];
    if (child == parent) continue;
    min_id = min(min_id, dfs_find_min(child, x));
  }
  return min_id;
}
```

1140 国王的遗产 代码

主DFS过程

```
int total;
int pivot;
SubTree best;
void dfs(int x, int parent) {
  subtree[x].num_nodes = 1;
  subtree[x].min_id = x;
  for (int i = 0; i < G[x].size(); ++i) {
    int child = G[x][i];
    if (child == parent) continue;
    dfs(child, x);
    subtree[x].num_nodes += subtree[child].num_nodes;
    subtree[x].min_id = min(subtree[x].min_id, subtree[child].min_id);
  }
  if (x != pivot) {
    subtree[x].outsider = parent;
    subtree[x].insider = x;
    if (subtree[x].num_nodes <= total / 2) {
      best = min(best, subtree[x]);
    }
    SubTree subtree2;
    subtree2.num_nodes = total - subtree[x].num_nodes;
    subtree2.min_id = pivot;
    subtree2.insider = parent;
    subtree2.outsider = x;
    if (subtree2.num_nodes <= total / 2) {
      best = min(best, subtree2);
    }
  }
}
```

1438 Shopaholic

1438 Shopaholic 题目大意

买东西 每买三件东西 最便宜的一件免费

给出n个需要买的东西

问最多免费多少？

$n \leq 20000, price \leq 20000$

1438 Shopaholic 样例

6

400 100 200 350 300 250

(400 350 300) (250 200 100) => 400

1438 Shopaholic 解题思路

尽量使价格高的东西免费

按价格从高到低排序，每三件取一件免费

严格证明？考虑最便宜的商品，必须找两个和它配对

```
sort(price, price + n, greater<int>());
for (int i = 2; i < n; i += 3) {
  saved += price[i];
}
```

1028 Hanoi Tower Sequence

1028 Hanoi Tower Sequence 题目大意

汉诺塔：三个柱子和大小两两不同的盘子放在一个柱子上

目的：将这些盘子移动到另外一个柱子上

规则：每次只能移动最顶端的一个盘子，每次移动后较小的盘子必须放在较大的盘子上面

现在给出步数p，问第p步移动的盘子的大小

$p \leq 10^{100}$

1028 Hanoi Tower Sequence 题目大意

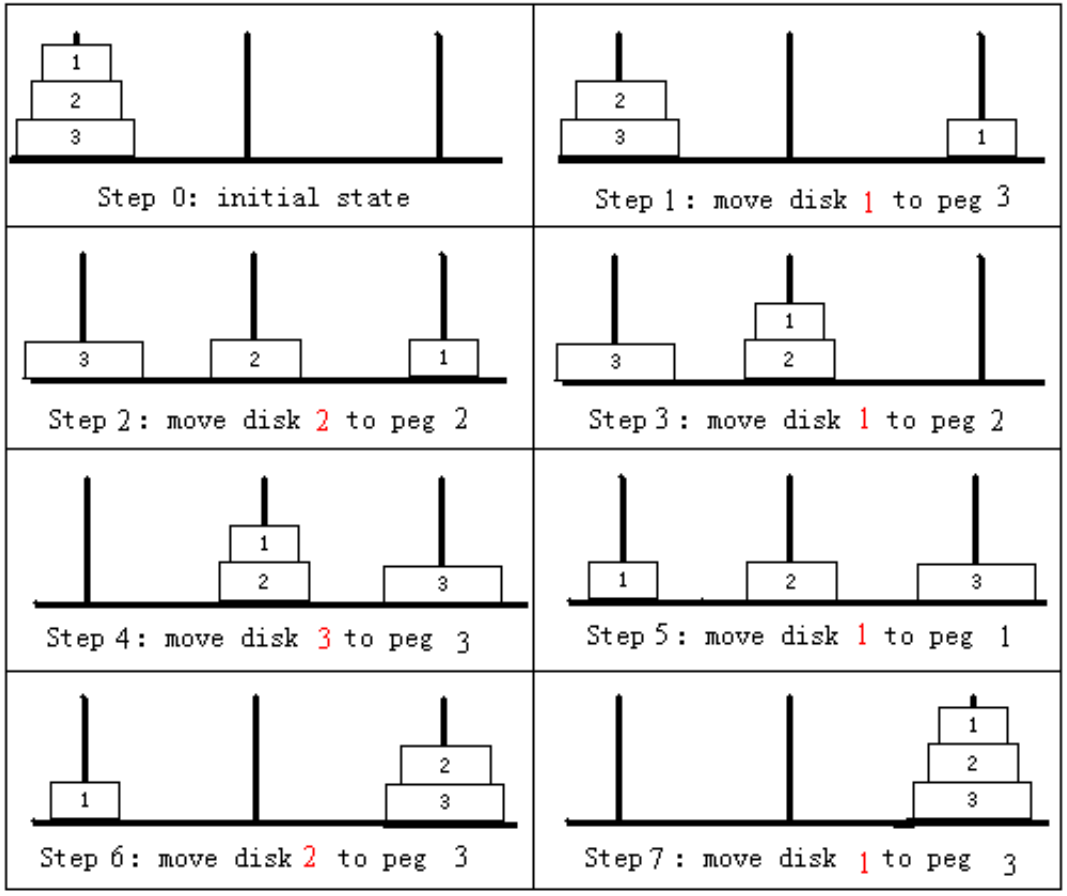


Fig 1: Demo of moving 3 disks from peg 1 to peg 3.

规则

1. 把前n - 1个盘子移到第二根柱子上

2. 把第n个盘子移到第三根柱子上
3. 把前n - 1个盘子移到第三根柱子上

1028 Hanoi Tower Sequence 解题思路

先递归求解总步数

设移动k个盘子需要f(k)步，则有：

f(k) = f(k - 1) + 1 + f(k - 1)

f(1) = 1

所以：

$f(k) = 2^k - 1$

1028 Hanoi Tower Sequence 解题思路

发现：

第2^k步移动的是第k + 1个盘子

1028 Hanoi Tower Sequence 解题思路

找规律

```
1
1 2 1
1 2 1 3 1 2 1
1 2 1 3 1 2 1 4 1 2 1 3 1 2 1
1 2 1 3 1 2 1 4 1 2 1 3 1 2 1 5 1 2 1 3 1 2 1 4 1 2 1 3 1 2 1
```

具有自相似的特性

```
0001 => 0
0010 => 1
0011 => 0
0100 => 2
0101 => 0
0110 => 1
0111 => 0
1000 => 3
1001 => 0
1010 => 1
1011 => 0
1100 => 2
1101 => 0
1110 => 1
1111 => 0
```

1028 Hanoi Tower Sequence 解题思路

答案为把p表示成2进制数，最后有多少个零，再加1

假设x是读进来的数字，x[n - 1]是最低位：

```
int ans = 1;
while (x[n - 1] % 2 == 0) {
  ++ans;

  // 除2
  int remain = 0;
  for (int i = 0; i < n; ++i) {
    remain = remain * 10 + x[i];
    x[i] = remain / 2;
    remain %= 2;
  }
  assert(remain == 0);
}
```

1029 Rabbit

1029 Rabbit 题目大意

开始有一对成年兔子

每对成年兔子每个月产生一对小兔子

每只小兔子经过m个月变成成年兔子

问经过d个月后有多少兔子

$1 \leq m \leq 10, 1 \leq d \leq 100$

1029 Rabbit 题目大意

m = 2时是经典的Fibonacci问题

1029 Rabbit 解题思路

每个月的兔子数量 = 上个月兔子数量 + 这个月出生的小兔子数量

小兔子由大兔子生育得到，这些大兔子在m个月前就必须存在了

每个月的兔子数量 = 上个月兔子数量 + m个月前的兔子数量

```
dp[n] = dp[n - 1] + dp[n - m]
```

1029 Rabbit 代码

```
struct BigInteger {
  static const int kMaxLen = 100;
  int x[kMaxLen];
  BigInteger(int a = 0) {
    memset(x, 0, sizeof (x));
    x[0] = a;
  }
};

// 效率较低，只做演示用！
BigInteger operator+(const BigInteger& a, const BigInteger& b) {
  BigInteger c;
  for (int i = 0; i < kMaxLen - 1; ++i) {
    c.x[i] += a.x[i] + b.x[i];
    c.x[i + 1] += c.x[i] / 10;
    c.x[i] %= 10;
  }
  return c;
}

dp[0] = BigInteger(1);
for (int i = 1; i <= d; ++i) {
  if (i < m) {
    a[i] = a[i - 1] + BigInteger(1);
  } else {
    a[i] = a[i - 1] + a[i - m];
  }
}
```

1381 a*b

1381 a*b 题目大意

给两个整数a和b，求$a \times b$

$0 \leq a \leq 10^{100}, 0 \leq b \leq 10000$

a, b都有可能是0

1381 a*b 解题思路

高精度乘法，模拟竖式乘法

输出时注意前导0和0的情况

1381 a*b 代码

```
struct BigInteger {
  static const int kMaxLen = 120;
  int x[kMaxLen];
  BigInteger(int a = 0) {
    memset(x, 0, sizeof (x));
    x[0] = a;
  }
  BigInteger operator*(int b) const {
    BigInteger res(0);
    for (int i = 0; i < kMaxLen; i++) {
      res.x[i] += x[i] * b;
      res.x[i + 1] += c.x[i] / 10;
      res.x[i] %= 10;
    }
    return res;
  }
};
```

以此为基础，可自行练习大整数乘大整数的程序。

另外，这两个大整数程序都以10为进位， 实际使用时可以用10000为进位数，把4个数字压在一个数组中，可以明显提高程序效率。

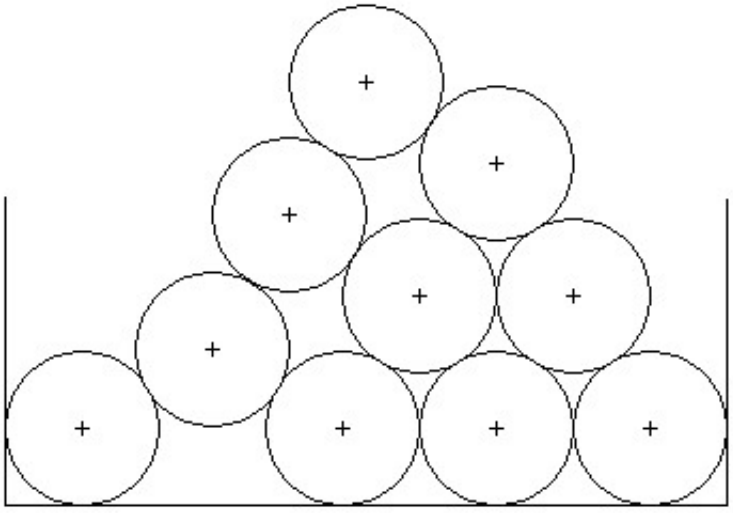
1206 1012 Stacking Cylinders

1206 1012 Stacking Cylinders 题目大意

给出最底层的n个圆柱的位置，求最顶层的圆柱的位置

圆柱半径都为1

$1 \leq n \leq 10$



如图

1206 1012 Stacking Cylinders 解题思路

已知两个圆的圆心坐标，求放在这两个圆上的圆的圆心坐标

向量 勾股定理

1206 1012 Stacking Cylinders complex库

STL中的复数库可以简化代码

复数的旋转：$a \times e^{i\theta}$

两个点之间的距离：$\|a - b\|$

```
complex<double> a, b;
b = a * exp(complex<double>(0, t));  //旋转角度为t
b = a * complex<double>(0, 1);  //旋转90度
sqrt(norm(a - b));  // 两点之间的距离
abs(a - b);  // 或者
a.real();  // x坐标
a.imag();  // y坐标
```

1206 1012 Stacking Cylinders 代码

```
typedef complex<double> Point;

bool cmp(const Point& a, const Point& b) {
  return a.real() < b.real();
}

Point Calculate(const Point& a, const Point& b) {
  Point mid = (a + b) / Point(2, 0);
  Point height = (b - mid) * Point(0, 1);
  double len = sqrt(4 - norm(a - mid));
  height = height / abs(height) * len;
  return mid + height;
}

sort(points, points + n, cmp);
for (int len = n - 1; len >= 1; --len) {
  for (int i = 0; i < len; ++i) {
    points[i] = Calculate(points[i], points[i + 1]);
  }
}
```

1172 Queens, Knights and Pawns

1172 Queens, Knights and Pawns 题目大意

给一个棋盘，若干后、马和兵的位置

求棋盘上有多个没被占领的格子不会受到后也不会受到马的攻击

棋盘大小1000×1000，每种棋子最多100个

1172 Queens, Knights and Pawns 解题思路

用二维数组表示一个棋盘

标记每个棋子的位置

再标记每个棋子能攻击的位置

最后计算有多少个位置不会被攻击

1172 Queens, Knights and Pawns 代码

```
enum GridState {
  empty,
  occupied,
  attacked
};

const int kMaxN = 1024;
GridState board[kMaxN][kMaxN];

void occupy(vector<Point> v) {
  for (int i = 0; i < v.size(); i++) {
    grid[v[i].x][v[i].y]] = occupied;
  }
}

bool in_board_and_unoccupied(Point p) {
  if (1 <= p.x && p.x <= num_row) {
    if (1 <= p.y && p.y <= num_col) {
      return grid[p.x][p.y]!=occupied;
    }
  }
  return false;
}
```

1172 Queens, Knights and Pawns 代码

```
int dKnight[8][2] = {{1,2},{1,-2},{2,-1},{-2,-1},{-1,-2},{-1,2},{-2,1},{2,1}};
int dQueen[8][2] = {{1,0},{1,-1},{0,-1},{-1,-1},{-1,0},{-1,1},{0,1},{1,1}};

void KnightAttack(vector<Point> points) {
  for (int i = 0; i < points.size(); i++) {
    for (int dir = 0; dir < 8; dir++) {
      Point newp(points[i].x + dKnight[dir][0], points[i].y + dKnight[dir][1]);
      if (in_board_and_unoccupied(newp)) {
        grid[newp.x][newp.y] = attacked;
      }
    }
  }
}
void QueenAttack(vector<Point> points) {
  for (int i = 0; i < points.size(); i++) {
    for (int dir = 0; dir < 8; dir++) {
      Point newp(points[i].x + dQueen[dir][0], points[i].y + dQueen[dir][1]);
      if (in_board_and_unoccupied(newp)) {
        grid[newp.x][newp.y] = attacked;
      }
    }
  }
}
```

1172 Queens, Knights and Pawns 代码

```
memset(grid, 0, sizeof(grid));
occupy(queen);
occupy(knight);
occupy(pawn);

KnightAttack(knight);
QueenAttack(queen);

int ans = 0;
for (int i = 1; i <= num_row; i++) {
  for (int j = 1; j <= num_col; j++) {
    if (grid[i][j] == empty) {
      ans++;
    }
  }
}
```

1034 Forest

1034 Forest 题目大意

n个节点

m条有向边

判断是否组成森林，如果是，求出它的最大深度和最大宽度

$n, m \leq 100$