36. 递推与递归

36.1 递归

36.1.1 递归实现指数型枚举 (5 s)

题意

从$1\sim n$这n $(1\leq n\leq 15)$个数中任意选取若干个,输出所有可能的选择方案.每种方案输出一行,每行内的数升序排列,对选0个数的方案输出空行.SPJ,不同方案间的顺序任意.

代码

```cpp
const int MAXN = 16;
int n;
int state[MAXN];  // 记录每个数的状态,0表示未考虑,1表示选,2表示不选

void dfs(int pos) {
  if (pos > n) {
    for (int i = 1; i <= n; i++)
      if (state[i] == 1) cout << i << ' ';
    cout << endl;
    return;
  }

  // 该位置不选
  state[pos] = 2;
  dfs(pos + 1);
  state[pos] = 0;

  // 该位置选
  state[pos] = 1;
  dfs(pos + 1);
  state[pos] = 0;
}

int main() {
  cin >> n;

  dfs(1);
}
```

36.1.2 递归实现排列型枚举 (5 s)

题意

按字典序升序输出$1\sim n$ $(1\leq n\leq 9)$的全排列.

思路

DFS,枚举每个位置填的数.

搜索树共n层.根节点有n个子节点,第2层的每个节点有$(n-1)$个子节点,即2层共$n(n-1)$个节点,$\cdots$,第n层共$n!$个节点.

树中节点每个节点都需枚举n个数,叶子节点每个节点需输出方案,故每个节点的时间复杂度都为$O(n)$.

总时间复杂度$n\left[1+n+n(n-1)+n(n-1)(n-2)+\cdots+n!\right]=n\cdot S$,显然$S\geq n!$.

$S=n!+\dfrac{n!}{1}+\dfrac{n!}{1\cdot 2}+\cdots+\dfrac{n!}{(n-1)!}+\dfrac{n!}{n!}\leq n!\left(1+1+\dfrac{1}{2}+\dfrac{1}{4}+\cdots\right)\leq 3n!$,故总时间复杂度$O(n\cdot n!)$.

代码

```cpp
const int MAXN = 15;
int n;
int state[MAXN];  // 记录每个位置放的数,0表示未放
bool used[MAXN];  // 记录每个数是否被用过

void dfs(int pos) {
  if (pos > n) {
    for (int i = 1; i <= n; i++) cout << state[i] << ' ';
    cout << endl;
    return;
  }

  for (int i = 1; i <= n; i++) {
    if (!used[i]) {
      state[pos] = i, used[i] = true;
      dfs(pos + 1);
      state[pos] = 0, used[i] = false;
    }
  }
}

int main() {
  cin >> n;

  dfs(1);
}
```

36.1.3 递归实现组合型枚举

题意

从$1\sim n$的n $(n>0)$个整数中随机选m $(0\leq m\leq n, n+(n-m)\leq 25)$个数,按字典序输出所有方案,每行输出一个方案,同个方案内的数升序排列.

思路

DFS枚举时规定选择的数按从小到大排序,只需保证所有相邻的两数都是后一个比前一个大.

因$C_n^m = C_n^{n-m}$,则组合数最大值C_{18}^7,计算知不会超时.

剪枝:因需保证选择的数升序,若将比当前位置填的数大的所有数都选上也不足m个树,则该分支无解.设当前位置为pos,当前位置可填的数的最小值为$start$,则$pos-1+(n-start+1)<m$,即$pos+m-start<m$时无解

代码

```cpp
const int MAXN = 30;
int n, m;  // C(n,m)
int state[MAXN];  // 记录每个位置填的数,0表示未填

void dfs(int pos, int start) {  // 当前填的位置、当前位置可填的数的最小值
  if (pos + n - start < m) return;  // 剪枝

  if (pos > m) {
    for (int i = 1; i <= m; i++) cout << state[i] << ' ';
    cout << endl;
    return;
  }

  for (int i = start; i <= n; i++) {
    state[pos] = i;
    dfs(pos + 1, i + 1);
    state[pos] = 0;
  }
}

int main() {
  cin >> n >> m;

  dfs(1, 1);  // 从第1个位置填1开始枚举
}
```

36.1.4 a-Good String

原题指路:https://codeforces.com/contest/1385/problem/D

题意

对长度为n $(n=2^k,k\in\mathbb{N})$的字符串$s[1\cdots n]$,称其为$c-$好串,如果下列条件中至少有一个条件满足:①长度为1,且包含字符c,即$s_1=c$;②长度>1,前一半字符串$s\left[1,\frac{n}{2}\right]$只含字符$c$,且后一半字符串$s\left[\frac{n}{2}+1,n\right]$是$(c+1)-$好串;③长度$>1$,后一半字符串$s\left[1,\frac{n}{2}\right]$只含字符$c$,且前一半字符串$s\left[\frac{n}{2}+1,n\right]$是$(c+1)-$好串.

每次操作可将s的一个字符变为其他小写字母.给定字符串s,求将其变为$a-$好串的最小操作次数.

有t $(1\le t\le 2e4)$组测试数据.每组测试数据第一行输入一个整数n $(1\le n\le 131072=2^{17})$,表示字符串$s$的长度.第二行输入长度为$n$且仅包含小写字母的字符串$s$.数据保证所有测试数据的$n$之和不超过$2e5$.

思路

每次递归到左右半边子串,将其变为$a-$好串.递归终止条件为条件①,答案为条件②和③的情况取$\min$.

代码

```cpp
const int MAXN = 2e5 + 5;
char s[MAXN];

int cal(int l, int r, char ch) {  // 求将子串str[l...r]变为ch-好串的最小操作次数
  if (l == r) return s[l] != ch;

  int mid = l + r >> 1;
  int lcnt = 0, rcnt = 0;
  for (int i = l; i <= mid; i++) lcnt += s[i] != ch;
  for (int i = mid + 1; i <= r; i++) rcnt += s[i] != ch;
  return min(lcnt + cal(mid + 1, r, ch + 1), rcnt + cal(l, mid, ch + 1));
}

void solve() {
  int n; cin >> n;
  cin >> s + 1;
  cout << cal(1, n, 'a') << endl;
}

int main() {
  CaseT  // 单测时注释掉该行
  solve();
}
```

36.1.5 Continued Fraction

原题指路:https://codeforces.com/gym/103366/problem/B

题意

给定一个既约分数$\frac{x}{y}$,将其化为连分数的形式$a_0+\cfrac{1}{a_1+\cfrac{1}{a_2+\cfrac{1}{\ddots+\cfrac{1}{a_n}}}}$.

有$t\ \ (1\le t\le 1000)$组测试数据.每组测试数据输入两个整数$x,y\ \ (1\le x,y\le 1\text{e}9,\gcd(x,y)=1)$.

对每组测试数据,先输出$n\ \ (0\le n\le 100)$,再输出$a_0,\cdots,a_n\ \ (0\le a_i\le 1\text{e}9)$.若有多组解,输出任一组.

思路

①若$y\mid x$,即$\frac{x}{y}$是整数时,它已是连分数.

②若$\frac{x}{y}$是假分数,提出$a_0=\left\lfloor\frac{x}{y}\right\rfloor$即可变为真分数,故只需考虑真分数的情况.

③若$\frac{x}{y}$是真分数,注意到$\frac{x}{y}=\frac{1}{\frac{y}{x}}$,其中$\frac{y}{x}$是假分数,递归处理即可.递归终止条件:$\frac{x}{y}$是整数.

事实上,②和③可统一为$\frac{x}{y}=\left\lfloor\frac{x}{y}\right\rfloor+\frac{x \bmod y}{y}=\left\lfloor\frac{x}{y}\right\rfloor+\frac{1}{\frac{y}{x \bmod y}}$.

该过程类似于辗转相除法,时间复杂度$O(\log \max\{x,y\})$.

代码

```cpp
void cal(int x, int y, vi& res) {
  res.push_back(x / y);
  if (x % y == 0) return;  // 递归终止条件

  cal(y, x % y, res);
}

void solve() {
  int x, y; cin >> x >> y;

  vi ans;
  cal(x, y, ans);

  cout << ans.size() - 1 << ' ';  // 注意-1
  for (auto i : ans) cout << i << ' ';
  cout << endl;
}

int main() {
  CaseT  // 单测时注释掉该行
  solve();
}
```

36.2 递推

36.2.1 简单斐波那契

题意

输入整数n $(0<n<46)$,输出斐波那契数列的前n项.如$n=5$时,前5项为:0 1 1 2 3.

代码I

```cpp
const int MAXN = 50;
int n;
int fib[MAXN];

int main() {
  cin >> n;

  fib[1] = 0, fib[2] = 1;
  for (int i = 3; i <= n; i++) fib[i] = fib[i - 1] + fib[i - 2];

  for (int i = 1; i <= n; i++) cout << fib[i] << ' ';
}
```

思路

注意到$fib[n]\ \ (n \geq 3)$只与$fib[n-1]$和$fib[n-2]$有关,则可用变量$a = fib[n-2], b = fib[n-1]$,下一步令$a = fib[n-1], b = fib[n]$,滚动.

代码II:滚动优化空间

```cpp
int main() {
  int n; cin >> n;

  int a = 0, b = 1;
  for (int i = 0; i < n; i++) {
    cout << a << ' ';
    int c = a + b;
    a = b, b = c;
  }
}
```

36.2.5 费解的开关

题意

25盏灯排成5×5的正方形,每盏灯有一开关,玩家可改变其状态.每一步,玩家可选择改变某一灯的状态,同时该灯上下左右的灯的状态也会改变.用1表示灯亮,0表示灯灭.给定游戏初始状态,判断6步内是否可使所有灯亮.

有$t\ \ (1 \leq t \leq 500)$组测试数据,每组测试数据输入5×5的矩阵表示初始状态,各组测试数据间用空行分隔.

若能在6步内使所有灯亮,输出最小步数,否则输出-1.

思路

注意到最终每个开关的状态只与其及其上下左右的切换次数的奇偶有关,则最终结果与操作顺序无关.注意到每个灯至多切换一次状态.

枚举第一行的$2^5 = 32$种操作,考虑第二行的操作.注意到此时能影响第一行的灯的状态的只有其正下方的第二行的灯,为使所有灯亮,则第一行灭的灯的下方的灯必须切换,第一行亮的灯的下方的灯必须不切换,即第二行的操作被第一行灯的状态唯一确定.同理下一行的状态被当前行的状态唯一确定.操作完第5行后,第$1 \sim 5$行灯全亮,若此时第6行灯全亮,则有解,更新最小步数;若此时第6行灯不全亮,则无解.

代码

```cpp
const int MAXN = 6;
char graph[MAXN][MAXN], backup[MAXN];
int dx[5] = { -1,0,1,0,0 }, dy[5] = { 0,1,0,-1,0 };

void turn(int x, int y) {
  for (int i = 0; i < 5; i++) {
    int curx = x + dx[i], cury = y + dy[i];
    if (curx < 0 || curx >= 5 && cury < 0 || cury >= 5) continue;
    graph[curx][cury] ^= 1;
  }
}

int main() {
```

```
  CaseT{
    for (int i = 0; i < 5; i++) cin >> graph[i];

    int ans = INF;  // >6的数
    for (int op = 0; op < 32; op++) {  // 枚举第一行的状态
      memcpy(backup, graph, so(graph));
      int step = 0;
      for (int i = 0; i < 5; i++) {
        if (op >> i & 1) {
          step++;
          turn(0, i);
        }
      }

      for (int i = 0; i < 4; i++) {  // 下一行的操作由当前行完全确定
        for (int j = 0; j < 5; j++) {
          if (graph[i][j] == '0') {
            step++;
            turn(i + 1, j);
          }
        }
      }

      bool ok = true;
      for (int i = 0; i < 5; i++) {
        if (graph[4][i] == '0') {
          ok = false;
          break;
        }
      }

      if (ok) ans = min(ans, step);
      memcpy(graph, backup, so(graph));
    }

    if (ans > 6) ans = -1;
    cout << ans << endl;
  }
}
```

36.2.6 带分数

题意

100可表示为带分数的形式$100 = 3 + \frac{69258}{714} = 82 + \frac{3546}{197}$,其中带分数中$1 \sim 9$每个数字恰出现一次.类似这样的带分数,100有11种表示法.输入正整数n $(1 \leq n < 1e6)$,输出方案数.

思路

$n = a + \frac{b}{c}$,其中a, b, c的数码中$1 \sim 9$恰出现一次.

暴力做法:枚举$1 \sim 9$的全排列,时间复杂度$O(n \cdot n!)$,再枚举将哪段区间作为a, b, c,即枚举隔板的位置,时间复杂度C_8^2,最后判断等式是否成立.总时间复杂度约$9e7$,但常数较小,可过.

考虑优化.注意到$1 \leq n \leq 1e6$,则$1 \leq a \leq 1e6$.又$cn = ac + b$,则可对每个枚举的a,再枚举c,最后计算出b.实现时,将a的搜索树的叶子节点扩展成一棵c的搜索树.

代码

```cpp
const int MAXN = 10;
int n;
bool vis[MAXN];  // 记录每个数字是否被用过
int ans;

bool check(int a, int c) {
  ll b = (ll)n * c - (ll)a * c;
  if (!a || !b || !c) return false;

  static bool backup[MAXN];
  memcpy(backup, vis, so(vis));
  while (b) {
    int tmp = b % 10; b /= 10;
    if (!tmp || backup[tmp]) return false;
    backup[tmp] = true;
  }

  for (int i = 1; i <= 9; i++)
    if (!backup[i]) return false;
  return true;
}

void dfs_c(int used, int a, int c) {  // 当前用了几个数字、a的值、当前c的值
  if (used > 9) return;

  if (check(a, c)) ans++;

  for (int i = 1; i <= 9; i++) {  // 枚举c的下一位
    if (!vis[i]) {
      vis[i] = true;
      dfs_c(used + 1, a, c * 10 + i);
      vis[i] = false;
    }
  }
}

void dfs_a(int used, int a) {  // 当前用了几个数字、当前a的值
  if (a >= n) return;  // a=n时b=0

  if (a) dfs_c(used, a, 0);  // a非零时,对每个a枚举c

  for (int i = 1; i <= 9; i++) {  // 枚举a的下一位
    if (!vis[i]) {
      vis[i] = true;
      dfs_a(used + 1, a * 10 + i);
      vis[i] = false;
    }
  }
}

```

```cpp
int main() {
  cin >> n;

  dfs_a(0, 0);  // 当前用了0个数,当前a的值为0

  cout << ans;
}
```

36.3.7 飞行员兄弟

题意

16个开关排成4×4的矩阵,每个开关有闭合和断开两种状态,分别用$+$和$-$表示.每一步可切换开关(i,j) $(1 \le i,j \le 4)$的状态,同时第i行和第j列的开关状态都切换.求将所有开关断开的最小切换步数.

输入4×4的矩阵表示开关初始状态,数据保证至少有一开关为闭合状态.

第一行输出最小切换步数n,接下来n行输出方案,每行输出两个数i,j $(1 \le i,j \le 4)$表示切换开关(i,j)的状态.若存在多种开冰箱的方式,则按照优先级整体从上到下、同行从左到右打开.

思路

暴力枚举$2^{16} = 65536$种方案,有16个开关,每个开关会改变7个开关的状态,检查16个开关是否都断开,若是,扫一遍记录方案,总时间复杂度$2^{16}(16 \times 7 + 16 + 16) \approx 1e7$.

因要输出字典序最小的方案,当两方案的最小步数相等时,应先枚举二进制数中1更靠前(低位)的方案.

代码I

```cpp
const int MAXN = 5;
char graph[MAXN][MAXN], backup[MAXN][MAXN];

int get(int x, int y) { return x * 4 + y; }

void turn_one(int x, int y) {
  graph[x][y] = graph[x][y] == '+' ? '-' : '+';
}

void turn_all(int x, int y) {
  for (int i = 0; i < 4; i++) turn_one(x, i), turn_one(i, y);
  turn_one(x, y);  // (x,y)被改变两次
}

int main() {
  for (int i = 0; i < 4; i++) cin >> graph[i];

  vii ans;
  for (int op = 0; op < 1 << 16; op++) {  // 枚举每个开关的操作
    vii tmp;
    memcpy(backup, graph, so(backup));

    for (int i = 0; i < 4; i++) {
      for (int j = 0; j < 4; j++) {
        if (op >> get(i, j) & 1) {
          tmp.push_back({ i,j });
          turn_all(i, j);
```

```
        }
      }
    }

    bool ok = true;
    for (int i = 0; ok && i < 4; i++) {
      for (int j = 0; j < 4; j++) {
        if (graph[i][j] == '+') {
          ok = false;
          break;
        }
      }
    }

    if (ok)
      if (ans.empty() || ans.size() > tmp.size()) ans = tmp;

    memcpy(graph, backup, so(graph));
  }

  cout << ans.size() << endl;
  for (auto item : ans) cout << item.first + 1 << ' ' << item.second + 1 << endl;
}
```

代码II:位运算优化

```
const int MAXN = 5;
int graph[MAXN][MAXN];

int get(int x, int y) { return x * MAXN + y; }

int main() {
  for (int i = 0; i < 4; i++) {
    for (int j = 0; j < 4; j++) {
      for (int k = 0; k < 4; k++)
        graph[i][j] += (1 << get(i, k)) + (1 << get(k, j));
      graph[i][j] -= 1 << get(i, j);
    }
  }

  int state = 0;
  for (int i = 0; i < 4; i++) {
    string line; cin >> line;
    for (int j = 0; j < 4; j++)
      if (line[j] == '+') state += 1 << get(i, j);
  }

  vii ans;
  for (int op = 0; op < 1 << 16; op++) {  // 枚举每个开关的操作
    int cur = state;
    vii tmp;

    for (int j = 0; j < 16; j++) {
      if (op >> j & 1) {
        int x = j / 4, y = j % 4;
        cur ^= graph[x][y];
```

```
        tmp.push_back({ x,y });
      }
    }

    if (!cur && (ans.empty() || ans.size() > tmp.size())) ans = tmp;
  }

  cout << ans.size() << endl;
  for (auto item : ans) cout << item.first + 1 << ' ' << item.second + 1 << endl;
}
```

36.3.8 翻硬币

题意

给定硬币的初始状态和目标状态,用$*$表示正面,o表示反面.每一步能翻转相邻两硬币,求最小翻转步数.

表示状态的字符串长度不超过100,数据保证一定有解.

思路

显然至多需50步.若用BFS,时间复杂度50^{99},会TLE.

将硬币的正反看作灯泡的亮灭,在相邻两灯泡间放一个可反转它们的开关.从左往右考察初始状态与目标状态的当前位置是否相同,若不同,则必须按开关,否则必须不按开关.显然解是被唯一确定的.总时间复杂度$O(n)$.

代码

```
const int MAXN = 105;
string start, target;

void turn(int pos) {
  start[pos] = start[pos] == '*' ? 'o' : '*';
}

int main() {
  cin >> start >> target;

  int ans = 0;
  for (int i = 0; i < start.length() - 1; i++) {
    if (start[i] != target[i]) {
      turn(i), turn(i + 1);
      ans++;
    }
  }
  cout << ans;
}
```

36.2.9 约数之和

题意

给定自然数a,b $(0\le a,b\le 5e7,a和b不同时为0)$,设$a^b$的所有约数之和为$s$.求$s \bmod 9001$.

思路

对$n=p_1^{\alpha_1}p_2^{\alpha_2}\cdots p_k^{\alpha_k}$,约数个数为$(\alpha_1+1)(\alpha_2+1)\cdots(\alpha_k+1)$,

约数之和为$(1+p_1+p_1^2+\cdots+p_1^{\alpha_1})(1+p_2+p_2^2+\cdots+p_2^{\alpha_2})\cdots(1+p_k+p_k^2+\cdots+p_k^{\alpha_k})$.

设$a=p_1^{\alpha_1}p_2^{\alpha_2}\cdots p_k^{\alpha_k}$.则$a^b=p_1^{b\alpha_1}p_2^{b\alpha_2}\cdots p_k^{b\alpha_k}$,代入约数之和公式计算,其中括号内的项可用等比数列求和公式.

考察如何用分治求出$sum(p,k)=p^0+p^1+\cdots+p^{k-1}$.

①k为偶数时,$sum(p,k)=\left(p^0+p^1+\cdots+p^{\frac{k}{2}-1}\right)+\left(p^{\frac{k}{2}}+p^{\frac{k}{2}+1}+\cdots+p^{k-1}\right)$

$=sum\left(p,\frac{k}{2}\right)+p^{\frac{k}{2}}\cdot sum\left(p,\frac{k}{2}\right)=\left(1+p^{\frac{k}{2}}\right)sum\left(p,\frac{k}{2}\right)$.

②k为奇数时,

$sum(p,k)=p^0+p^1+\cdots+p^{k-1}=p^0+p\left(p^0+p^1+\cdots+p^{k-2}\right)=1+p\cdot sum(p,k-1)$

$=sum(p,k-1)+p^{k-1}$.

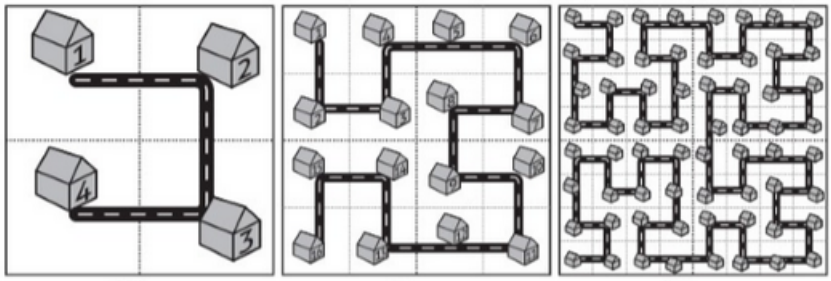
最坏每两次操作有一次可以项数可以除以2,时间复杂度$O(2\log n)$,再乘上快速幂$O(\log n)$,总时间复杂度$O(\log^2 n)$.

代码

```cpp
const int MOD = 9901;

ll sum(int p, int k) {
  if (k == 1) return 1;
  else if (k & 1) return (sum(p, k - 1) + qpow(p, k - 1, MOD)) % MOD;
  else return (1 + qpow(p, k / 2, MOD)) * sum(p, k / 2) % MOD;
}

int main() {
  int a, b; cin >> a >> b;

  int ans = 1;
  for (int i = 2; i <= a / i; i++) {  // 对a分解素因数
    if (a % i == 0) {
      int s = 0;
      while (a % i == 0) a /= i, s++;
      ans = (ll)ans * sum(i, b * s + 1) % MOD;  // 注意bs+1
    }
  }
  if (a > 1) ans = (ll)ans * sum(a, b + 1) % MOD;  // 大于√a的素因子

  if (!a) ans = 0;  // 0的非0次方为0
  cout << ans;
}
```

36.2.10 分形之城

题意



如上图是一个城市规划方案.城市规模扩大后,将与原城区结构一样的区域按上图所示的方式建在原城市的周围,提高城市等级.

对任意等级的城市,将正方形街区从左上角开始按道路编号.设每个街区都是边长为10的正方形.求等级为n的城市中编号为a和b的两街区间的直线距离(即街区中心点间的距离),答案四舍五入取整.

有t $(1 \le t \le 1000)$组测试数据.每组测试数据输入三个整数n, a, b $(1 \le n \le 31, 1 \le a, b \le 2^{2n})$.

思路

设街区编号从0开始,顺时针编号$0, 1, 2, \cdots$.在每个等级的城市中,以左上角0号街区为原点,向下建立x轴,向右建立y轴.

考察如何在等级为n $(n > 1)$的城市中求编号为a的街区的坐标,记函数$get(n, a)$.显然等级为n的城市的边长为2^n,每个等级的城市可分为4个小正方形,每个正方形的边长为$len = 2^{n-1}$,每个小正方形中有$block = 2^{2(n-1)}$个街道,则a所在的小正方形的编号即$\left\lfloor \frac{a}{2^{2n-2}} \right\rfloor$,递归到对应的小正方形即可,注意递归时$a$的编号改变,显然递归到$get(n-1, a\%block)$.求得$a$在$(n-1)$级的城市中的坐标$(x, y)$后,根据$a$在第$n$级的城市中所在的小正方形的编号,给对应的坐标关于$x, y$轴对称或加上偏移量$len$即可.

时间复杂度取决于递归层数,为$O(n)$.

代码

```cpp
struct Point { ll x, y; };

Point get(int n, ll a) {
  if (!n) return { 0,0 };

  ll block = (ll)1 << n * 2 - 2;  // 每个小正方形中的街区数
  ll len = (ll)1 << n - 1;  // 每个小正方形的边长

  auto p = get(n - 1, a % block);
  ll x = p.x, y = p.y;

  int idx = a / block;  // 街区a在等级为n的城市中所在的小正方形的编号
  switch (idx) {
    case 0: return { y,x }; break;
    case 0: return { x,y + len }; break;
    case 2: return { x + len, y + len }; break;
    case 3: return { len * 2 - 1 - y,len - 1 - x }; break;
  }
}

```

```
int main() {
  CaseT{
    int n; ll a,b; cin >> n >> a >> b;
    auto pa = get(n, a - 1), pb = get(n, b - 1);  // 注意街区编号从0开始
    cout << (ll)(10 * hypot(pa.x - pb.x, pa.y - pb.y) + 0.5) << endl;
  }
}
```