算法分析与设计
第二章

1020383827@qq.com

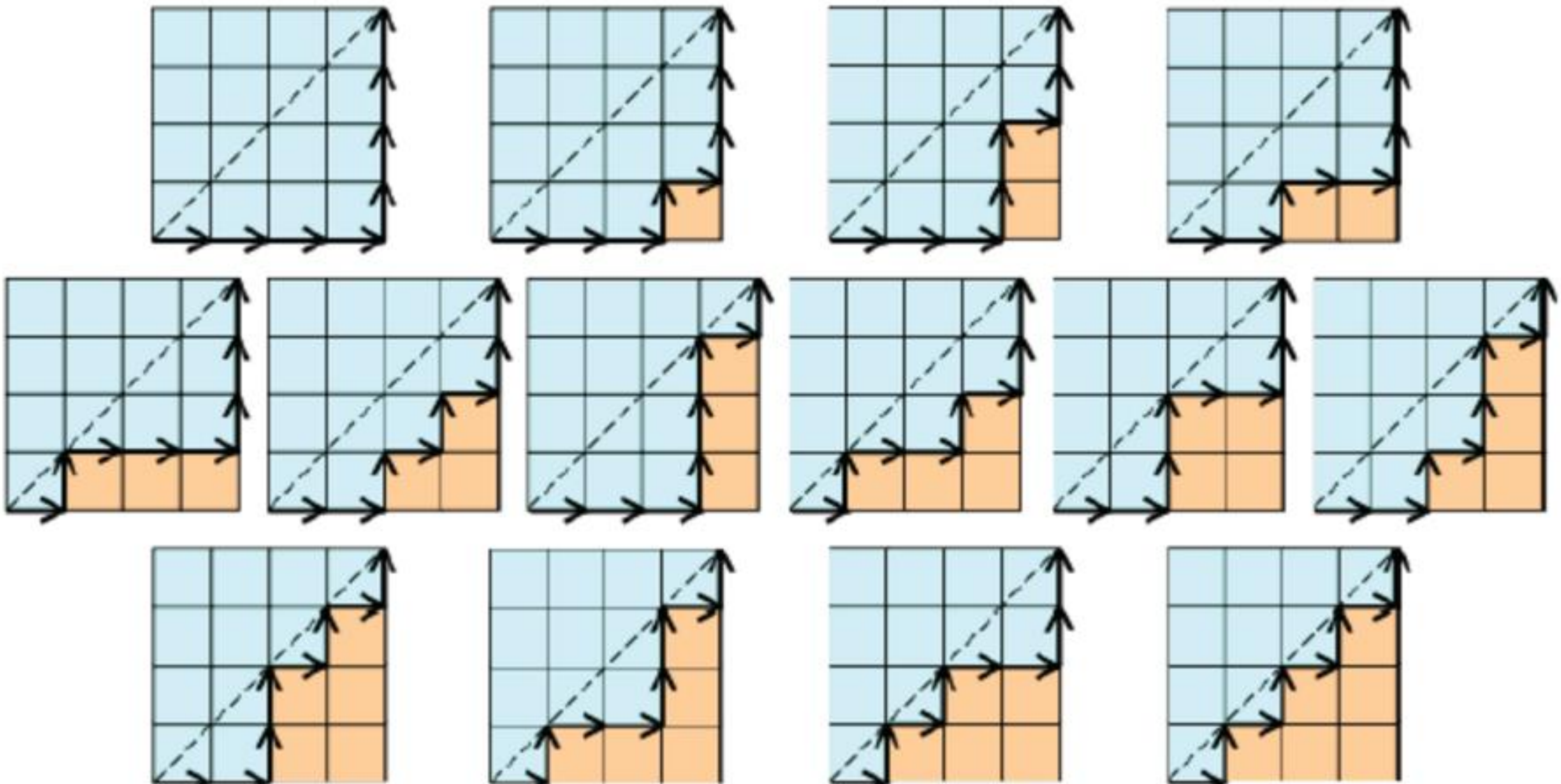
doubleh

- 13062
- 1011
- 1345
- 1176
- 1121
- 1264
- 1828
- 1527
- 1148
- 1163
- 1687

soj 13062 SubDiagonal Paths

- 题意：给出n，在n*n的格子中，一开始你位于(0,0)，然后你每次可以从(x,y)走到(x+1,y)，或者(x,y+1)，全程需满足x>=y，问这样走到(n,n)处有几种走法。
- 题解：设dp[i][j]表示从(0,0)走到(i,j)的走法，那么就有dp[i][j] = dp[i][j-1] + dp[i-1][j]，并且注意边界条件为dp[0][j]=1,dp[i][j]=0(i>j)
- 所求答案即为卡特兰数 (by poetry)

soj 13062 SubDiagonal Paths



Soj 1011 Lenny's Lucky Lotto

- 题意：
- 给出N，M，问有多少个长度为N的整数序列，满足所有数都在[1,M]内，并且每一个数至少是前一个数的两倍
- 限制：
- 1<=N<=10
- 1<=M<=2000
- Eg. N = 4, M = 10, 4种情况
- [1, 2, 4, 8], [1, 2, 4, 9], [1, 2, 4, 10], [1, 2, 5, 10]

Soj 1011 Lenny's Lucky Lotto

- 解法:
- 动态规划
- f[i][j]表示序列长度为i，最后一个数为j的序列数
- 那么，可以通过枚举倒数第二个数求得
- $f[i][j] = \sum_{k=1}^{j/2} f[i-1][k]$
- 复杂度为：O(N*M^2)

```
memset(dp,0,sizeof(dp));
for(int i=1;i<=m;i++)
    dp[1][i]=1;
for(int i=2;i<=n;i++)
{
    for(int j=i;j<=m;j++)
    {
        for(int k=i-1;k<=j/2;k++)
        {
            dp[i][j]+=dp[i-1][k];
        }
    }
}
```

Soj 1011 Lenny's Lucky Lotto

- 解法：
- 其实递推式可以优化，注意到递推式是个前缀和
- 我们不妨另记s[i][j]表示序列长度为i，最后一个数不超过j的序列数
- 那么显然s[i][j] = s[i][j – 1] + f[i][j]，而f[i][j] = s[i-1][j/2]
- 时间复杂度下降为：O(N*M)

Soj 1011 Lenny's Lucky Lotto

- 代码

```
void dp()
{
    for (int j = 1; j < maxm; j++)
    {
        f[1][j] = 1;
        s[1][j] = j;
    }
    for (int i = 2; i < maxn; i++)
    {
        f[i][0] = s[i][0] = 0;
        for (int j = 1; j < maxm; j++)
        {
            f[i][j] = s[i - 1][j / 2];
            s[i][j] = s[i][j - 1] + f[i][j];
        }
    }
}
```

Soj 1345 能量项链

- 题意：
- 给出一串项链，每次可以选相邻两个珠子进行聚合，释放出一定的能
- 量，并产生一个新珠子
- 项链是头尾相接的
- 求释放的能量的总和的最大值
- 限制：
- 项链长度不超过100

Soj 1345 能量项链

- 题意：

例如：设N=4，4颗珠子的头标记与尾标记依次为(2，3) (3，5) (5，10) (10，2)。我们用记号⊕表示两颗珠子的聚合操作，(j⊕k)表示第j，k两颗珠子聚合后所释放的能量。则第4、1两颗珠子聚合后释放的能量为：

(4⊕1)=10*2*3=60。

这一串项链可以得到最优值的一个聚合顺序所释放的总能量为

((4⊕1)⊕2)⊕3) =10*2*3+10*3*5+10*5*10=710。

Soj 1345 能量项链

- 解法：
- 同样动态规划
- f[i][j]表示合并i~j这段数字能够获得的最大能量
- 转移时，枚举i~k与k+1~j这两段合并，获得的能量就为w[i]*w[k+1]*w[j+1]

- 注意到这是个环，所以应用破环为链的方法，扩展两倍

Soj 1345 能量项链

- 代码：
- 也可以写成记忆化

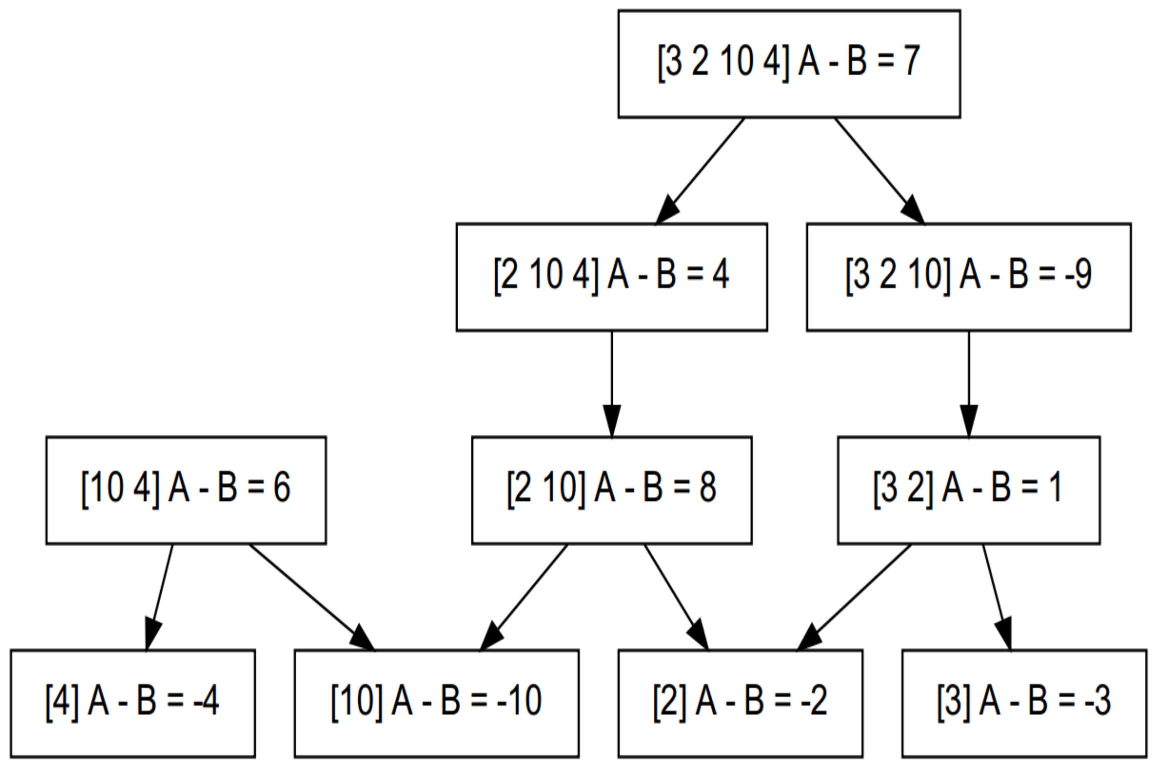
```
int solve()
{
    for (int i = 0; i < 2 * n; i++)
        f[i][i] = 0;
    for (int len = 2; len <= n; len++)
    {
        for (int i = 0; i + len <= 2 * n; i++)
        {
            int j = i + len - 1;
            f[i][j] = 0;
            for (int k = i; k < j; k++)
            {
                int tmp = f[i][k] + f[k + 1][j];
                tmp += w[i % n] * w[(k + 1) % n] * w[(j + 1) % n];
                f[i][j] = max(f[i][j], tmp);
            }
        }
    }
    int ret = 0;
    for (int i = 0; i < n; i++)
        ret = max(ret, f[i][i + n - 1]);
    return ret;
}
```

soj 1176 Two Ends

- 题意：给一个长度为n的数列ai，两个人在上面做交替取数，每个人的每一轮从这个数列的两端中取出一个数（不能不操作）。先手可以自由选择策略，后手选择贪心策略。贪心策略是指，如果两端数大小不同，选择大的那个；如果相同选择左边那个。问最后先手能赢后手多少分。
- 约束：1 <=n <= 1000且n为偶数，sum ai <= 1,000,000

soj 1176 Two Ends

- 解法1：搜索
- 例子
- [3 2 10 4]



soj 1176 Two Ends

- 解法2：记忆化搜索，动态规划
- 核心：对于已经计算过的状态不再计算，记忆化下来。
- 本质：动态规划

soj 1176 Two Ends

- 解法2：记忆化搜索，动态规划
- 参考代码：

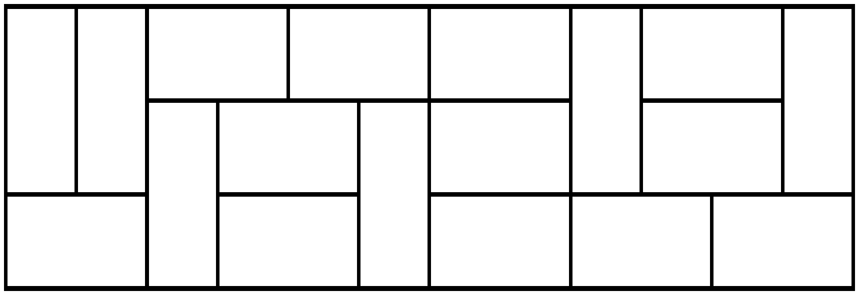
```
const int maxn = 1010;
int n;
bool cal[maxn][maxn];
int a[maxn];
int f[maxn][maxn];
int dp(int l, int r)
{
    if (l > r) return 0;
    if (cal[l][r]) return f[l][r];
    cal[l][r] = true;
    f[l][r] = 0;

    // take left
    f[l][r] = max(f[l][r], a[l + 1] >= a[r]? dp(l + 2, r) + a[l] - a[l + 1]: dp(l + 1, r - 1) + a[l] - a[r]);
    // take right
    f[l][r] = max(f[l][r], a[l] >= a[r - 1]? dp(l + 1, r - 1) + a[r] - a[l]: dp(l, r - 2) + a[r] - a[r - 1]);

    return f[l][r];
}
int twoEnd()
{
    for (int i = 0; i < n; i++) for (int j = 0; j < n; j++)
        cal[i][j] = false;
    return dp(0, n - 1);
}
```

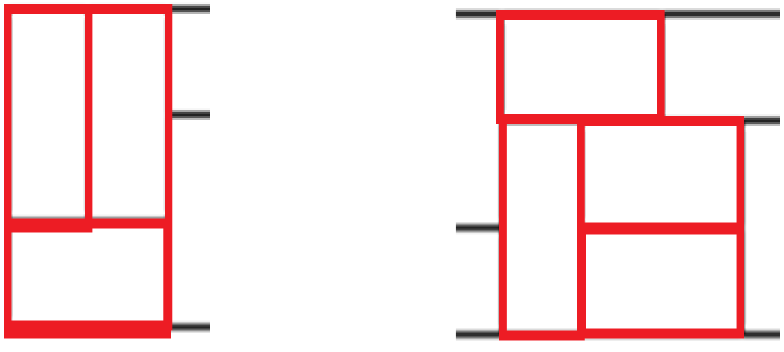
Soj 1121 Tri Tiling

- 题意：
- 用1*2的长方形铺满3*n的长方形，有多少种方法
- 限制：
- 1<=N<=30



Soj 1121 Tri Tiling

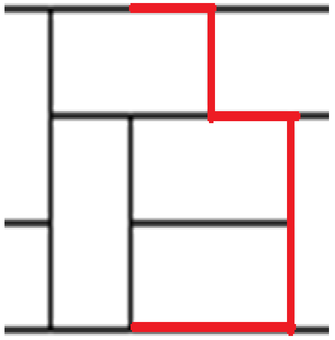
- 分析：
- n为偶数时注意到有两种单元，上下翻转看成同一类



Soj 1121 Tri Tiling

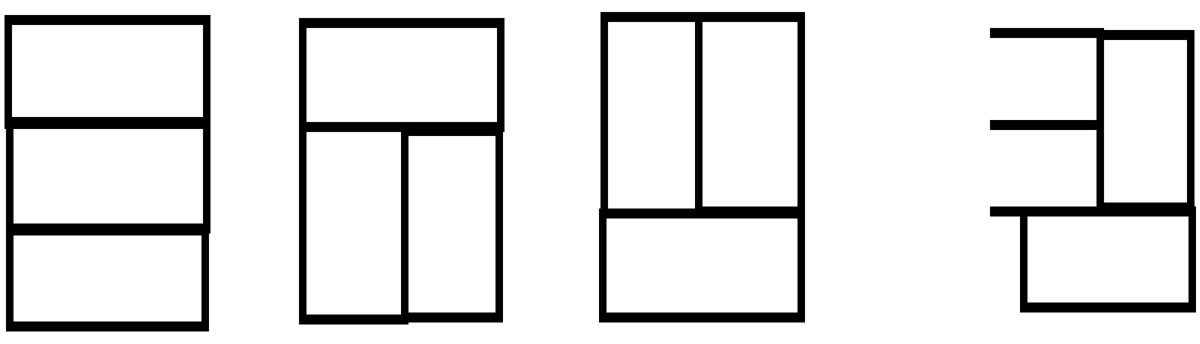
- 分析：
- 令f[i]表示铺满3*(2i)的长方形的方案数

- 令g[i]表示铺满3*(2i)的长方形，并且向后伸展出如同下图轮廓的方案数



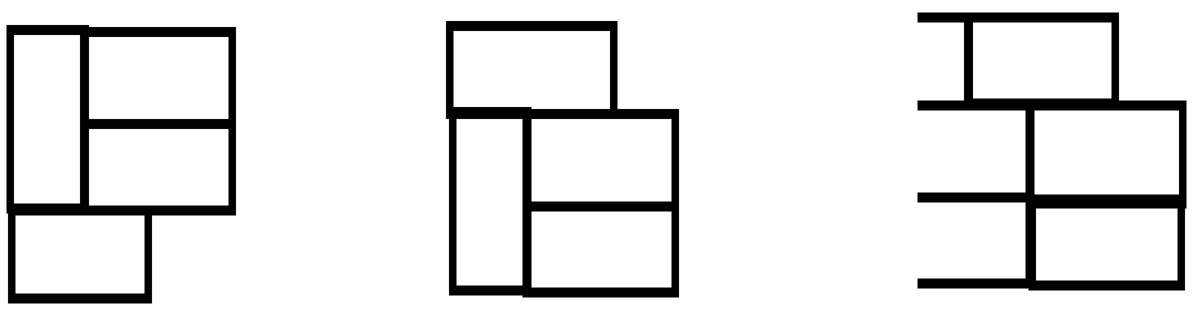
Soj 1121 Tri Tiling

- 分析：
- 那么有：
- f[i]=3*f[i-1]+g[i-1]



Soj 1121 Tri Tiling

- 分析：
- 那么有：
- g[i]=2*f[i-1]+g[i-1]



Soj 1121 Tri Tiling

- 代码：
- 最后答案为f[n/2]

```
const int maxn = 16;
int f[maxn];
int g[maxn];
void tritiling()
{
    f[0] = 1;
    g[0] = 0;
    for (int i = 1; i < maxn; i++)
    {
        f[i] = 3 * f[i - 1] + g[i - 1];
        g[i] = 2 * f[i - 1] + g[i - 1];
    }
}
```

Soj 1264 Atomic Car Race

- 题意：
- 在一次赛车比赛中，共有n个检查点，在每个检查点可以选择是否更换轮胎，更换轮胎需要花费b单位时间
- 在一次更换轮胎之后，赛车的速度先增加（轮胎变热），后减少（轮胎损坏），每单位公里行驶需要的时间可以表达为（x为离最近更换轮胎距离，从x公里跑到x+1公里）：
- 1/(v - e * (x - r)) (if x >= r)
- 1/(v - f * (r - x)) (if x < r)
- 现在问从起点到终点的最少时间
- 限制：
- 0<n<=100， 0<=r<=an-1，0<a1<a2<…<an<=10000， v – e * (an – 1 r)>=0.01，v - f * r >= 0.01

Soj 1264 Atomic Car Race

- 解法：
- 动态规划，我们很容易发现如果确定最后一次在哪里换轮胎，之前在哪里换轮胎是不会影响后面的过程
- 那么f[i]表示在最后一次在第i个检查点换轮胎
- 另外g[i]表示连续行驶i公里所需要的时间
- 对于f[i]，我们可以枚举上次换轮胎的检查点j得到
- $f[i] = \min\{f[j] + g[a[i] - a[j]]+b\}$
- 边界为$f[0] = 0$
- 答案为$f[n] - b$（最后一次不需要更换轮胎了）
- 时间复杂度：O(N^2+an)

Soj 1264 Atomic Car Race

- 代码：

```
const int maxn = 100;
const int maxm = 10001;
const double inf = 1e30;
int a[maxn], n;
double b, c, d, e, f, v, r;
double F[maxn];
double G[maxm];
double carRace()
{
    G[0] = 0;
    for (int i = 0; i < a[i]; i++)
        G[i + 1] = G[i] + (i < r? 1. / (v - f * (r - i)): 1. / (v - e * (i - r)));
    F[0] = 0;
    for (int i = 1; i <= n; i++)
    {
        F[i] = inf;
        for (int j = 0; j < i; j++)
            F[i] = min(F[i], F[j] + G[a[i] - a[j]] + b);
    }
    return F[n] - b;
}
```

Soj 1828 Minimal

- 题意：
- 给出两个集合S1和S2，
  在S2中选出一些不重复的数与S1的每个数匹配，使得匹配的数的差的
  绝对值之和尽量小
- 限制：
- 集合中数的个数不超过500

Soj 1828 Minimal

- 分析：
- 注意到假设我们从S2中选择了某些数，假设选出的数组成S3
- 那么S3中最小的数肯定匹配S1中最小的数，S3中次小的数匹配S1中次小的数…
- 证明：
- 假设S1中有了a1,a2,S3中有b1,b2
- 且a1<=a2,b1<=b2
- 那么有|a1-b1|+|a2-b2|<=|a1-b2|+|a2-b1|
- 所以从小到大一个个匹配不会更差，只会更好

Soj 1828 Minimal

- 解法：
- 动态规划。
- 对S1和S2都从小到大排序
- f[i][j]表示S1中的前i个数，与S2中的前j个数匹配（j个数选i个数出来）
- 那么显然有
- f[i][j] = inf, j < i
- f[0][0] = 0
- f[i][j] = min{f[i-1][j-1]+|a[i]-b[j]|,f[i][j-1]},j>=i

Soj 1828 Minimal

- 代码:

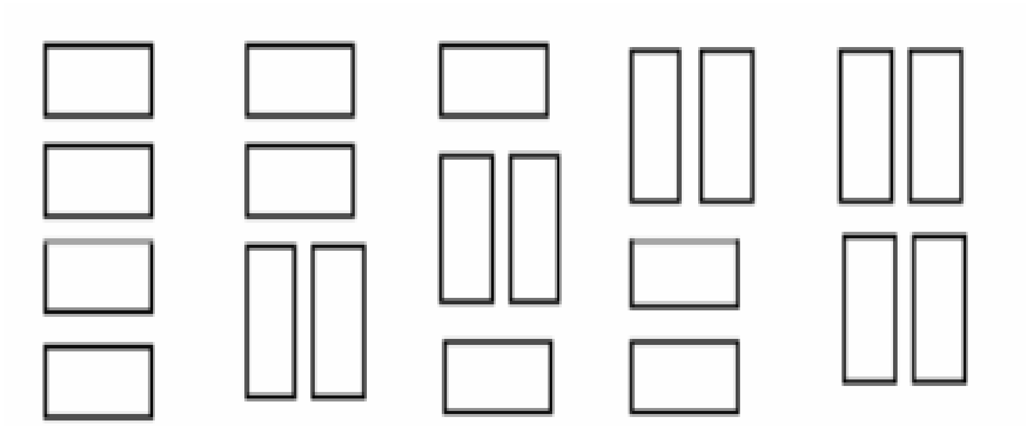
```
const int maxn = 501;
const int inf = 1<<30;
int n, m;
int a[maxn], b[maxn];
int f[maxn][maxn];
int minimal()
{
    sort(a + 1, a + n + 1);
    sort(b + 1, b + m + 1);
    for (int i = 0; i <= n; i++)
        for (int j = 0; j <= m; j++)
        {
            if (i == 0 && j == 0)
                f[i][j] = 0;
            else if (j < i)
                f[i][j] = inf;
            else
            {
                f[i][j] = inf;
                if (i > 0)
                    f[i][j] = min(f[i][j], f[i - 1][j - 1] + abs(a[i] - b[j]));
                if (j > 0)
                    f[i][j] = min(f[i][j], f[i][j - 1]);
            }
        }
    return f[n][m];
}
```

Soj 1527 Tiling a Grid With Dominoes

- 题意：
- 题目类似前面的1121，现在用骨牌填充4*n的矩形，问有多少方案
- 限制：
- 最终答案在32位整数范围内

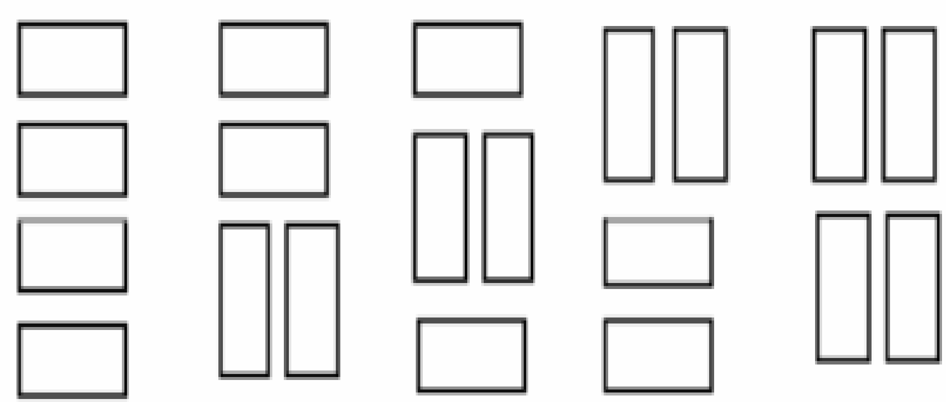
Soj 1527 Tiling a Grid With Dominoes

- 解法一：
- 继续用之前的方法，分析出所有基本构成



Soj 1527 Tiling a Grid With Dominoes

- 解法一：
- 其实我们发现只有轮廓才是我们关注的
- 对于宽为4的矩形，要完全填满其实只有以下几种情况
- f[i][j]表示前4*i列已经完全填满，并且第(i+1)列情况为第j种的方案数
- j=0:0000
- j=1:1100
- j=2:0011
- j=3:1001
- j=4:0110



Soj 1527 Tiling a Grid With Dominoes

- 解法一：
- j=0:0000
- j=1:1100
- j=2:0011
- j=3:1001
- j=4:0110
- 转移:
- f[i][0] = f[i-1][0] + f[i-1][1] + f[i-1][2] + f[i-1][3] + f[i-2][0]
- f[i][1] = f[i-1][0] + f[i-1][2]
- f[i][2] = f[i-1][0] + f[i-1][1]
- f[i][3] = f[i-1][0] + f[i-1][4]
- f[i][4] = f[i-1][3]

Soj 1527 Tiling a Grid With Dominoes

- 代码：

```
void pre()
{
    f[0][0] = 1;
    for (int i = 1; i < maxn; i++)
    {
        f[i][0] = f[i - 1][0] + f[i - 1][1] + f[i - 1][2] + f[i - 1][3];
        if (i > 1) f[i][0] += f[i - 2][0];
        f[i][1] = f[i - 1][0] + f[i - 1][2];
        f[i][2] = f[i - 1][0] + f[i - 1][1];
        f[i][3] = f[i - 1][0] + f[i - 1][4];
        f[i][4] = f[i - 1][3];
    }
}
```

Soj 1527 Tiling a Grid With Dominoes

- 解法二：
- 状态压缩dp
- 这里直接用4位0/1表示轮廓，总共2^4种状态
- 如0101虽然在合法解不可能出现但仍然表示出来，就是这么暴力

- 这种方法较难，建议课下学习

Soj 1148 过河

- 题意：
- 桥的起点为0，终点为L，其中地有M个石子
- 青蛙每次跳的范围为[S,T]，问要跳过桥最小踩到石子次数
- 限制：
- 1<=L<=10^9
- 1<=S<=T<=10
- 1<=M<=100

Soj 1148 过河

- 分析：
- 如果L不是那么大，那么直接动态规划大家都知道怎么做
- 注意到S和T不大于10，我们就从这个入手

Soj 1148 过河

- 分析：
- 考虑从0出发，那么如果能够到达编号为n
- 我们可以用一个动态规划来“打表”
- 观察规律可以发现如果s<t，当n比较大时，总是可达的（大约为90，不妨设更大，100）

- 那么我们可以减少格子数，也就是只两个相邻石子之间只保留最多100个格子
- 这个时候我们可以发现，这时总格子数大约在1万左右，再应用动态规划足以解决问题

Soj 1148 过河

- 代码：
- 特判情况

```
if (S == T)
{
    int ret = 0;
    for (int i = 0; i < M; i++)
        if (pos[i] % S == 0)
            ret++;
    return ret;
}
```

Soj 1148 过河

- 代码：
- 预处理格子

```
}
sort(pos, pos + M);
N = 0;
memset(has, false, sizeof(has));
memset(f, -1, sizeof(f));
for (int i = 0, pre = 0; i < M; i++)
{
    N += min(pos[i] - pre, 100);
    has[N] = true;
    pre = pos[i];
}
N += min(L - pos[M - 1], 100);
```

Soj 1148 过河

- 代码：
- 动态规划

```
f[0] = 0;
for (int i = 0; i < N; i++) if (f[i] >= 0)
{
    for (int j = S; j <= T; j++)
    {
        int k = i + j;
        if (f[k] < 0 || f[k] > f[i] + has[k])
            f[k] = f[i] + has[k];
    }
}

int ret = M;
for (int i = N; i < N + T; i++)
    ret = min(ret, f[i]);
return ret;
```

Soj 1148 过河

- 完整代码：
- http://soj.sysu.edu.cn/viewsource.php?sid=4467035

Soj 1163 Tour

- 题意：
- 旅行商问题的变种
- 一个想走一个环路，经过N个地点，并且先从左往右走，再从右往左走
- 求最短路程
- 限制：
- N题目没给出，但是可以认为1<=N<=50
- 坐标的x值均不相同

Soj 1163 Tour

- 分析：
- 注意到x坐标各不相同，这其实可以把原来的环路，当作两条从左到右的路径组成，同样从最左端走到最右端
- 那么我们可以使用f[i][j]表示两条路径一条从最左端走到i，并且另一条从最左端走到j的最短路程，并且不妨i<j

Soj 1163 Tour

- 代码：

```
double solve()
{
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            d[i][j] = hypot(x[i] - x[j], y[i] - y[j]);
    f[0][1] = d[0][1];
    for (int k = 2; k < n; k++)
    {
        for (int i = 0; i < k - 1; i++)
            f[i][k] = f[i][k - 1] + d[k - 1][k];
        f[k - 1][k] = inf;
        for (int i = 0; i < k - 1; i++)
            f[k - 1][k] = min(f[k - 1][k], f[i][k - 1] + d[i][k]);
    }
    return f[n - 2][n - 1] + d[n - 2][n - 1];
}
```

Soj 1687 Permutation

- 题意：
- n个数的排列，可以在中间插入小于号和大于号
- 如1 3 5 4 2 变成1<3<5>4>2
- 现在问n个数其中有k个小于号的排列有多少个
- 限制：
- 1<=n,k<=100

Soj 1687 Permutation

- 解法：
- 再从枚举的角度上看，从小到大的插入每个数
- 假设我们现在枚举到i
- 这个时候它肯定是最大的一个数
- 我们可以根据插入到位置分类：
- 如果插入到序列开头或者小于号中间，增加一个大于号；
- 如果插入到序列结尾或者大于号中间，增加一个小于号

Soj 1687 Permutation

- 解法：
- 动态规划，用f[i][j]表示长度为i，有j个小于号的序列数
- 那么
- f[i+1][j]+=f[i][j]*(j + 1)
- f[i+1][j+1]+=f[i][j]*(i-j)

Soj 1687 Permutation

- 代码：

```
void pre()
{
    f[1][0] = 1;
    for (int i = 1; i < 100; i++)
    {
        for (int j = 0; j < i; j++)
        {
            f[i + 1][j] += f[i][j] * (j + 1);
            f[i + 1][j] %= mod;
            f[i + 1][j + 1] += f[i][j] * (i - j);
            f[i + 1][j + 1] %= mod;
        }
    }
}
```