状态压缩动态规划（简称状压dp）是另一类非常典型的动态规划，通常使用在NP问题的小规模求解中，虽然是指数级别的复杂度，但速度比搜索快，其思想非常值得借鉴。

为了更好的理解状压dp，首先介绍位运算相关的知识。

1.’&’符号，x&y，会将两个十进制数在二进制下进行与运算，然后返回其十进制下的值。例如3(11)&2(10)=2(10)。

2.’|’符号，x|y，会将两个十进制数在二进制下进行或运算，然后返回其十进制下的值。例如3(11)|2(10)=3(11)。

3.’^’符号，x^y，会将两个十进制数在二进制下进行异或运算，然后返回其十进制下的值。例如3(11)^2(10)=1(01)。

4.’<<’符号，左移操作，x<<2，将x在二进制下的每一位向左移动两位，最右边用0填充，x<<2相当于让x乘以4。相应的，’>>’是右移操作，x>>1相当于给x/2，去掉x二进制下的最有一位。

这四种运算在状压dp中有着广泛的应用，常见的应用如下：

1.判断一个数字x二进制下第i位是不是等于1。

方法：if ( ( ( 1 << ( i - 1 ) ) & x ) > 0)

将1左移i-1位，相当于制造了一个只有第i位上是1，其他位上都是0的二进制数。然后与x做与运算，如果结果>0，说明x第i位上是1，反之则是0。

2.将一个数字x二进制下第i位更改成1。

方法：x = x | ( 1<<(i-1) )

证明方法与1类似，此处不再重复证明。

3.把一个数字二进制下最靠右的第一个1去掉。

方法：x=x&(x-1)

感兴趣的读者可以自行证明。

位运算在状压dp中用途十分广泛，请看下面的例题。

【例1】有一个N\*M(N<=5,M<=1000)的棋盘，现在有1\*2及2\*1的小木块无数个，要盖满整个棋盘，有多少种方式？答案只需要mod1,000,000,007即可。

例如：对于一个2\*2的棋盘，有两种方法，一种是使用2个1\*2的，一种是使用2个2\*1的。

【算法分析】

在这道题目中,N和M的范围本应该是一样的，但实际上，N和M的范围却差别甚远，对于这种题目，首先应该想到的就是，正确算法与这两个范围有关！N的范围特别小，因此可以考虑使用状态压缩动态规划的思想，请看下面的图：

假设第一列已经填满，则第二列的摆设方式，只与第一列对第二列的影响有关。同理，第三列的摆设方式也只与第二列对它的影响有关。那么，使用一个长度为N的二进制数state来表示这个影响，例如：4(00100)就表示了图上第二列的状态。

因此，本题的状态可以这样表示：

dp[i][state]表示该填充第i列，第i-1列对它的影响是state的时候的方法数。i<=M,0<=state<2N

对于每一列，情况数也有很多，但由于N很小，所以可以采取搜索的办法去处理。对于每一列，搜索所有可能的放木块的情况，并记录它对下一列的影响，之后更新状态。状态转移方程如下：

dp[i][state]=∑dp[i-1][pre]每一个pre可以通过填放成为state

对于每一列的深度优先搜索，写法如下：

**[cpp]** [view plain](http://blog.csdn.net/u011077606/article/details/43487421) [copy](http://blog.csdn.net/u011077606/article/details/43487421)

1. //第i列，枚举到了第j行，当前状态是state，对下一列的影响是nex
2. void dfs(int i,int j,int state,int nex)
3. {
4. if (j==N)
5. {
6. dp[i+1][nex]+=dp[i][state];
7. dp[i+1][nex]%=mod;
8. return;
9. }
10. //如果这个位置已经被上一列所占用,直接跳过
11. if (((1<<j)&state)>0)
12. dfs(i,j+1,state,nex);
13. //如果这个位置是空的，尝试放一个1\*2的
14. if (((1<<j)&state)==0)
15. dfs(i,j+1,state,nex|(1<<j));
16. //如果这个位置以及下一个位置都是空的，尝试放一个2\*1的
17. if (j+1<N && ((1<<j)&state)==0 && ((1<<(j+1))&state)==0)
18. dfs(i,j+2,state,nex);
19. return;
20. }

状态转移的方式如下：

**[cpp]** [view plain](http://blog.csdn.net/u011077606/article/details/43487421) [copy](http://blog.csdn.net/u011077606/article/details/43487421)

1. for (int i=1;i<=M;i++)
2. {
3. for (int j=0;j<(1<<N);j++)
4. if (dp[i][j])
5. {
6. dfs(i,0,j,0);
7. }
8. }

最终，答案就是dp[M+1][0]。

【代码实现】

**[cpp]** [view plain](http://blog.csdn.net/u011077606/article/details/43487421) [copy](http://blog.csdn.net/u011077606/article/details/43487421)

1. /\*
2. ID:aqx
3. PROG:铺地砖
4. LANG:c++
5. \*/
6. //第i列，枚举到了第j行，当前状态是state，对下一列的影响是nex
7. #include <cstdio>
8. #include <cstring>
9. #include <algorithm>
10. #include <iostream>
12. using namespace std;
14. int N, M;
15. long long dp[1005][34];
17. void dfs(int i,int j,int state,int nex)
18. {
19. if (j==N)
20. {
21. dp[i+1][nex]+=dp[i][state];
22. return;
23. }
24. //如果这个位置已经被上一列所占用,直接跳过
25. if (((1<<j)&state)>0)
26. dfs(i,j+1,state,nex);
27. //如果这个位置是空的，尝试放一个1\*2的
28. if (((1<<j)&state)==0)
29. dfs(i,j+1,state,nex|(1<<j));
30. //如果这个位置以及下一个位置都是空的，尝试放一个2\*1的
31. if (j+1<N && ((1<<j)&state)==0 && ((1<<(j+1))&state)==0)
32. dfs(i,j+2,state,nex);
33. return;
34. }
36. int main()
37. {
38. while (cin>>N>>M)
39. {
40. memset(dp,0,sizeof(dp));
41. if (N==0 && M==0) break;
42. dp[1][0]=1;
43. for (int i=1;i<=M;i++)
44. {
45. for (int j=0;j<(1<<N);j++)
46. if (dp[i][j])
47. {
48. dfs(i,0,j,0);
49. }
50. }
51. cout<<dp[M+1][0]<<endl;
52. }
53. }

【例2】最小总代价（Vijos-1456）

题目描述：

n个人在做传递物品的游戏,编号为1-n。

游戏规则是这样的：开始时物品可以在任意一人手上，他可把物品传递给其他人中的任意一位；下一个人可以传递给未接过物品的任意一人。

即物品只能经过同一个人一次，而且每次传递过程都有一个代价；不同的人传给不同的人的代价值之间没有联系；

求当物品经过所有n个人后，整个过程的总代价是多少。

输入格式：

第一行为n,表示共有n个人（16>=n>=2）；

以下为n\*n的矩阵，第i+1行、第j列表示物品从编号为i的人传递到编号为j的人所花费的代价，特别的有第i+1行、第i列为-1（因为物品不能自己传给自己），其他数据均为正整数(<=10000)。

(对于50%的数据，n<=11)。

输出格式：

一个数，为最小的代价总和。

输入样例：

2

-1 9794

2724 –1

输出样例：

2724

【算法分析】

看到2<=n<=16，应想到此题和状态压缩dp有关。每个人只能够被传递一次，因此使用一个n位二进制数state来表示每个人是否已经被访问过了。但这还不够，因为从这样的状态中，并不能清楚地知道现在物品在谁 的手中，因此，需要在此基础上再增加一个状态now，表示物品在谁的手上。

dp[state][now]表示每个人是否被传递的状态是state，物品在now的手上的时候，最小的总代价。

初始状态为：dp[1<<i][i]=0；表示一开始物品在i手中。

所求状态为：min(dp[(1<<n)-1][j]); 0<=j<n

状态转移方程是：

dp[state][now]=min(dp[pre][t]+dist[now][t])；

pre表示的是能够到达state这个状态的一个状态，t能够传递物品给now且只有二进制下第t位与state不同。

状态的大小是O((2n)\*n)，转移复杂度是O(n)。总的时间复杂度是O((2n)\*n\*n)。

【代码实现】

**[cpp]** [view plain](http://blog.csdn.net/u011077606/article/details/43487421) [copy](http://blog.csdn.net/u011077606/article/details/43487421)

1. /\*
2. ID:shijieyywd
3. PROG:Vijos-1456
4. LANG:c++
5. \*/
6. #include <cstdio>
7. #include <cstring>
8. #include <algorithm>
9. #include <vector>
11. #define MAXN 20
12. #define INF 0x3f3f3f3f
14. using namespace std;
16. int n;
17. int edges[MAXN][MAXN];
18. int dp[65546][MAXN];
20. int min(int a, int b)
21. {
22. if (a == -1) return b;
23. if (b == -1) return a;
24. return a < b ? a : b;
25. }
27. int main() {
28. freopen("p1456.in", "r", stdin);
29. scanf("%d", &n);
30. int t;
31. for (int i = 0; i < n; i++)
32. {
33. for (int j = 0; j < n; j++)
34. {
35. scanf("%d", &edges[i][j]);
36. }
37. }
38. memset(dp, -1, sizeof(dp));
39. for (int i = 0; i < n; i++)
40. {
41. dp[1 << i][i] = 0;
42. }
43. int ans = -1;
44. for (int i = 0; i < 1 << n; i++)
45. {
46. for (int j = 0; j < n; j++)
47. {
48. if (dp[i][j] != -1)
49. {
50. for (int k = 0; k < n; k++)
51. {
52. if (!(i & (1 << k)))
53. {
54. dp[i | (1 << k)][k] = min(dp[i | (1 << k)][k], dp[i][j] + edges[j][k]);
55. if ((i | (1 << k)) == (1 << n) - 1) ans = min(ans, dp[i | (1 << k)][k]);
56. }
57. }
58. }
59. }
60. }
61. if (ans != -1)
62. printf("%d\n", ans);
63. else printf("0\n");
65. return 0;
66. }

【例3】胜利大逃亡(续)(Hdoj-1429)

题目描述：

Ignatius再次被魔王抓走了(搞不懂他咋这么讨魔王喜欢)……

这次魔王汲取了上次的教训，把Ignatius关在一个n\*m的地牢里，并在地牢的某些地方安装了带锁的门，钥匙藏在地牢另外的某些地方。刚开始Ignatius被关在(sx,sy)的位置，离开地牢的门在(ex,ey)的位置。Ignatius每分钟只能从一个坐标走到相邻四个坐标中的其中一个。魔王每t分钟回地牢视察一次，若发现Ignatius不在原位置便把他拎回去。经过若干次的尝试，Ignatius已画出整个地牢的地图。现在请你帮他计算能否再次成功逃亡。只要在魔王下次视察之前走到出口就算离开地牢，如果魔王回来的时候刚好走到出口或还未到出口都算逃亡失败。

输入格式：

每组测试数据的第一行有三个整数n,m,t(2<=n,m<=20,t>0)。接下来的n行m列为地牢的地图，其中包括:

. 代表路

\* 代表墙

@ 代表Ignatius的起始位置

^ 代表地牢的出口

A-J 代表带锁的门,对应的钥匙分别为a-j

a-j 代表钥匙，对应的门分别为A-J

每组测试数据之间有一个空行。

输出格式：

针对每组测试数据，如果可以成功逃亡，请输出需要多少分钟才能离开，如果不能则输出-1。

输入样例：

4 5 17

@A.B.

a\*.\*.

\*..\*^

c..b\*

输出样例：

16

【算法分析】

初看此题感觉十分像是宽度优先搜索（BFS），但搜索的过程中如何表示钥匙的拥有情况，却是个问题。借鉴状态压缩的思想，使用一个10位的二进制数state来表示此刻对10把钥匙的拥有情况，那么，dp[x][y][state]表示到达(x,y)，钥匙拥有状况为state的最短路径。另外，需要注意到一旦拥有了某一把钥匙，那个有门的位置就如履平地了。

代码的实现方式可以采用Spfa求最短路的方式。值得一提的是，Spfa算法本来就是一种求解最短路径问题的动态规划算法，本文假设读者已经非常熟悉Spfa等基础算法，在此处不再赘述。

状态压缩dp可以出现在各种算法中，本题就是典型的搜索算法和状态压缩dp算法结合的题目。另外，很多状态压缩dp本身就是通过搜索算法实现的状态转移。

【代码实现】

**[cpp]** [view plain](http://blog.csdn.net/u011077606/article/details/43487421) [copy](http://blog.csdn.net/u011077606/article/details/43487421)

1. /\*
2. ID:shijieyywd
3. PROG:Hdu-1429
4. LANG:c++
5. \*/
6. #include <cstdio>
7. #include <cstring>
8. #include <algorithm>
9. #include <iostream>
10. #include <queue>
12. using namespace std;
14. struct Node{
15. int x;
16. int y;
17. int step;
18. int key;
19. Node() {}
20. Node(int a, int b, int s, int k) : x(a), y(b), step(s), key(k) {}
21. };
23. int n, m, t;
24. int arr[25][25];
25. int door[25][25];
26. int key[25][25];
27. int Go[4][2] = {{0, 1}, {0, -1}, {-1, 0}, {1, 0}};
28. int sx, sy;
29. int ex, ey;
30. int vis[25][25][1049];
32. bool canGo(int x, int y, int k)
33. {
34. if (x >= 0 && x < n && y >= 0 && y < m && !arr[x][y])
35. {
36. if (vis[x][y][k]) return false;
37. if ((k & door[x][y]) == door[x][y]) return true;
38. }
39. return false;
40. }
42. int bfs() {
43. memset(vis, 0, sizeof(vis));
44. queue<Node> q;
45. Node s = Node(sx, sy, 0, 0);
46. q.push(s);
47. vis[sx][sy][0] = 1;
48. while (!q.empty())
49. {
50. Node e = q.front();
51. q.pop();
52. if (e.x == ex && e.y == ey) return e.step;
53. for (int i = 0; i < 4; i++)
54. {
55. int nx = e.x + Go[i][0];
56. int ny = e.y + Go[i][1];
57. if (canGo(nx, ny, e.key))
58. {
59. Node nex = Node(nx, ny, e.step + 1, e.key | key[nx][ny]);
60. vis[nx][ny][nex.key] = 1;
61. q.push(nex);
62. }
63. }
64. }
65. return 0;
66. }
68. int main() {
69. while (~scanf("%d %d %d\n", &n, &m, &t))
70. {
71. memset(arr, 0, sizeof(arr));
72. memset(door, 0, sizeof(door));
73. memset(key, 0, sizeof(key));
74. char c;
75. for (int i = 0; i < n; i++)
76. {
77. for (int j = 0; j < m; j++)
78. {
79. scanf("%c", &c);
80. if (c == '\*') arr[i][j] = 1;
81. else if (c == '@') sx = i, sy = j;
82. else if (c == '^') ex = i, ey = j;
83. else if (c >= 'a' && c <= 'z') key[i][j] = 1 << (c - 'a');
84. else if (c >= 'A' && c <= 'Z') door[i][j] = 1 << (c - 'A');
85. }
86. getchar();
87. }
88. int ans = bfs();
89. if (ans < t && ans) printf("%d\n", ans);
90. else printf("-1\n");
91. }
92. return 0;
93. }