[状态压缩DP总结【POJ3254】【POJ1185】【POJ3311】【HDU3001】【POJ2288】【ZOJ4257】【POJ2411】【HDU3681】](http://blog.csdn.net/accry/article/details/6607703)

分类： [动态规划](http://blog.csdn.net/AcCry/article/category/764944)2011-07-15 10:10 10498人阅读 [评论](http://blog.csdn.net/accry/article/details/6607703#comments)(12) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/accry/article/details/6607703#report)

[vb](http://www.csdn.net/tag/vb)[path](http://www.csdn.net/tag/path)[forms](http://www.csdn.net/tag/forms)[c](http://www.csdn.net/tag/c)[struct](http://www.csdn.net/tag/struct)[each](http://www.csdn.net/tag/each)

动态规划本来就很抽象，状态的设定和状态的转移都不好把握，而状态压缩的动态规划解决的就是那种状态很多，不容易用一般的方法表示的动态规划问题，这个就更加的难于把握了。难点在于以下几个方面：状态怎么压缩？压缩后怎么表示？怎么转移？是否具有最优子结构？是否满足后效性？涉及到一些位运算的操作，虽然比较抽象，但本质还是动态规划。找准动态规划几个方面的问题，深刻理解动态规划的原理，开动脑筋思考问题。这才是掌握动态规划的关键。

动态规划最关键的要处理的问题就是位运算的操作，容易出错，状态的设计也直接决定了程序的效率，或者代码长短。状态转移方程一定要仔细推敲，不可一带而过，要思考为什么这么做，掌握一个套路，遇见这类问题能快速的识别出问题的本质，找出状态转移方程和DP的边界条件。

下面是一些关于状态压缩DP的题目，大都不难。状压DP的东西还有很多，还会接着总结。

[【POJ3254】Corn Fields](http://poj.org/problem?id=3254)

【题目大意】一个矩阵里有很多格子，每个格子有两种状态，可以放牧和不可以放牧，可以放牧用1表示，否则用0表示，在这块牧场放牛，要求两个相邻的方格不能同时放牛，即牛与牛不能相邻。问有多少种放牛方案(一头牛都不放也是一种方案)

【解析】根据题意，把每一行的状态用二进制的数表示，0代表不在这块放牛，1表示在这一块放牛。首先很容易看到，每一行的状态要符合牧场的硬件条件，即牛必须放在能放牧的方格上。这样就能排除一些状态。另外，牛与牛之间不能相邻，这样就要求每一行中不能存在两个相邻的1，这样也能排除很多状态。然后就是根据上一行的状态转移到当前行的状态的问题了。必须符合不能有两个1在同一列（两只牛也不能竖着相邻）的条件。这样也能去掉一些状态。然后，上一行的所有符合条件的状态的总的方案数就是当前行该状态的方案数。

【状态表示】dp[state][i]：在状态为state时，到第i行符合条件的可以放牛的方案数

【状态转移方程】dp[state][i] =Sigma dp[state'][i-1] （state'为符合条件的所有状态）

【DP边界条件】首行放牛的方案数dp[state][1] =1(state符合条件) OR 0 (state不符合条件)

【代码】

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/accry/article/details/6607703)

1. #include <cstdio>
2. #include <cstring>
3. **using** **namespace** std;
4. #define mod 100000000
5. **int** M,N,top = 0;
6. **int** state[600],num[110];
7. **int** dp[20][600];
8. **int** cur[20];
9. **inline** **bool** ok(**int** x){
10. **if**(x&x<<1)**return** 0;
11. **return** 1;
12. }
13. **void** init(){
14. top = 0;
15. **int** total = 1 << N;
16. **for**(**int** i = 0; i < total; ++i){
17. **if**(ok(i))state[++top] = i;
18. }
19. }
20. **inline** **bool** fit(**int** x,**int** k){
21. **if**(x&cur[k])**return** 0;
22. **return** 1;
23. }
24. **inline** **int** jcount(**int** x)
25. {
26. **int** cnt=0;
27. **while**(x)
28. {
29. cnt++;
30. x&=(x-1);
31. }
32. **return** cnt;
33. }
35. **int** main(){
36. **while**(scanf("%d%d",&M,&N)!= EOF){
37. init();
38. memset(dp,0,**sizeof**(dp));
39. **for**(**int** i = 1; i <= M; ++i){
40. cur[i] = 0;
41. **int** num;
42. **for**(**int** j = 1; j <= N; ++j){
43. scanf("%d",&num);
44. **if**(num == 0)cur[i] +=(1<<(N-j));
45. }
46. }
47. **for**(**int** i = 1;i <= top;i++){
48. **if**(fit(state[i],1)){
49. dp[1][i] = 1;
50. }
51. }
52. **for**(**int** i = 2; i <= M; ++i){
53. **for**(**int** k = 1; k <= top; ++k){
54. **if**(!fit(state[k],i))**continue**;
55. **for**(**int** j = 1; j <= top ;++j){
56. **if**(!fit(state[j],i-1))**continue**;
57. **if**(state[k]&state[j])**continue**;
58. dp[i][k] = (dp[i][k] +dp[i-1][j])%mod;
59. }
60. }
61. }
62. **int** ans = 0;
63. **for**(**int** i = 1; i <= top; ++i){
64. ans = (ans + dp[M][i])%mod;
65. }
66. printf("%d\n",ans);
67. }
68. }

[【POJ1185】炮兵阵地--经典状压DP](http://poj.org/problem?id=1185)

【题目大意】类似于上面一道题，一个方格组成的矩阵，每个方格可以放大炮用0表示，不可以放大炮用1表示（原题用字母），让放最多的大炮，大炮与大炮间不会互相攻击。

【解析】可以发现，对于每一行放大炮的状态，只与它上面一行和上上一行的状态有关，每一行用状态压缩的表示方法，0表示不放大炮，1表示放大炮，同样的，先要满足硬件条件，即有的地方不能放大炮，然后就是每一行中不能有两个1的距离小于2（保证横着不互相攻击），这些要预先处理一下。然后就是状态表示和转移的问题了，因为是和前两行的状态有关，所以要开个三维的数组来表示状态，当前行的状态可由前两行的状态转移而来。即如果当前行的状态符合前两行的约束条件（不和前两行的大炮互相攻击），则当前行的最大值就是上一个状态的值加上当前状态中1的个数（当前行放大炮的个数）

【状态表示】dp[i][j][k] 表示第i行状态为k，第i-1状态为j时的最大炮兵个数。

【状态转移方程】dp[i][k][t] =max(dp[i][k][t],dp[i-1][j][k]+num[t]); num[t]为t状态中1的个数

【DP边界条件】dp[1][1][i] =num[i] 状态i能够满足第一行的硬件条件（注意：这里的i指的是第i个状态，不是一个二进制数，开一个数组保存二进制状态）

【代码】

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/accry/article/details/6607703)

1. #include <cstdio>
2. #include <cstring>
3. **using** **namespace** std;
4. #define max(a,b) (a) > (b) ? (a) : (b)
6. **int** N,M;
7. **char** map[110][20],num[110],top;
8. **int** stk[70],cur[110];
9. **int** dp[110][70][70];
11. **inline** **bool** ok(**int** x){  //判断该状态是否合法，即不存在相邻的1之间的距离小于3的
12. **if**(x&(x<<1)) **return** 0;
13. **if**(x&(x<<2)) **return** 0;
14. **return** 1;
15. }
16. //找到所有可能的合法状态，最多60种
17. **inline** **void** jinit()
18. {
19. top=0;
20. **int** i,total=1<<N;
21. **for**(i=0;i<total;i++) **if**(ok(i)) stk[++top]=i;
22. }
23. //判断状态x是否与第k行匹配
24. **inline** **bool** fit(**int** x,**int** k)
25. {
26. **if**(cur[k]&x) **return** 0;
27. **return** 1;
28. }
29. //数一个整型数x的二进制中1的个数(用于初始化)
30. **inline** **int** jcount(**int** x)
31. {
32. **int** cnt=0;
33. **while**(x)
34. {
35. cnt++;
36. x&=(x-1);
37. }
38. **return** cnt;
39. }
41. **int** main(){
42. **while**(scanf("%d%d",&M,&N) != EOF){
43. **if**(N == 0 && M == 0)**break**;
44. jinit();
45. **for**(**int** i = 1; i <= M; ++i)scanf("%s",map[i]+1);
46. **for**(**int** i = 1; i <= M; ++i)
47. **for**(**int** j = 1; j <= N; ++j){
48. cur[i]=0;
49. **for**(j=1;j<=N;j++){
50. **if**(map[i][j]=='H')cur[i]+=(1<<(j-1));
51. }
52. }
53. memset(dp,-1,**sizeof**(dp));
54. //初始化第一行状态
55. **for**(**int** i = 1;i <= top;i++){
56. num[i]=jcount(stk[i]);
57. **if**(fit(stk[i],1))
58. dp[1][1][i]=num[i];
59. }
60. **int** i,t,j,k;
61. **for**(i = 2;i <= M;i++){
62. **for**(t = 1;t <= top;t++){
63. **if**(!fit(stk[t],i)) **continue**;
64. **for**(j = 1;j <= top;j++)
65. {
66. **if**(stk[t]&stk[j])**continue**;
67. **for**(k = 1;k <= top;k++)
68. {
69. **if**(stk[t]&stk[k])**continue**;
70. **if**(dp[i-1][j][k]==-1)**continue**;
71. dp[i][k][t] =max(dp[i][k][t],dp[i-1][j][k]+num[t]);
72. }
73. }
74. }
75. }
76. **int** ans = 0;
77. **for**(i = 1; i <= M; ++i)
78. **for**(j = 1; j <= top; ++j)
79. **for**(k = 1; k <= top; ++k)
80. ans = max(ans,dp[i][j][k]);
81. printf("%d\n",ans);
82. }
83. **return** 0;
84. }

[【POJ3311】Hie With The Pie](http://poj.org/problem?id=3311)

【题目大意】类似于TSP问题，只是每个点可以走多次，比经典TSP问题不同的是要先用弗洛伊的预处理一下两两之间的距离。求最短距离。

【解析】可以用全排列做，求出一个最短的距离即可。或者用状态压缩DP.用一个二进制数表示城市是否走过

【状态表示】dp[state][i]表示到达i点状态为state的最短距离

【状态转移方程】dp[state][i] =min{dp[state][i],dp[state'][j]+dis[j][i]} dis[j][i]为j到i的最短距离

【DP边界条件】dp[state][i] =dis[0][i]  state是只经过i的状态

【代码】

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/accry/article/details/6607703)

1. #include<iostream>
2. #define INF 100000000
3. **using** **namespace** std;
4. **int** dis[12][12];
5. **int** dp[1<<11][12];
6. **int** n,ans,\_min;
7. **int** main()
8. {
9. //freopen("in.txt","r",stdin);
10. **while**(scanf("%d",&n) && n)
11. {
12. **for**(**int** i = 0;i <= n;++i)
13. **for**(**int** j = 0;j <= n;++j)
14. scanf("%d",&dis[i][j]);
15. **for**(**int** k = 0;k <= n;++k)
16. **for**(**int** i = 0;i <= n;++i)
17. **for**(**int** j = 0;j <=n;++j)
18. **if**(dis[i][k] + dis[k][j]< dis[i][j])
19. dis[i][j] = dis[i][k] +dis[k][j];
21. **for**(**int** S = 0;S <= (1<<n)-1;++S)//枚举所有状态，用位运算表示
22. **for**(**int** i = 1;i <= n;++i)
23. {
24. **if**(S & (1<<(i-1)))//状态S中已经过城市i
25. {
26. **if**(S ==(1<<(i-1)))   dp[S][i] =dis[0][i];//状态S只经过城市I，最优解自然是从0出发到i的dis,这也是DP的边界
27. **else**//如果S有经过多个城市
28. {
29. dp[S][i] = INF;
30. **for**(**int** j = 1;j <=n;++j)
31. {
32. **if**(S &(1<<(j-1)) && j != i)//枚举不是城市I的其他城市
33. dp[S][i] =min(dp[S^(1<<(i-1))][j] + dis[j][i],dp[S][i]);
34. //在没经过城市I的状态中，寻找合适的中间点J使得距离更短
35. }
36. }
37. }
38. }
39. ans = dp[(1<<n)-1][1] + dis[1][0];
40. **for**(**int** i = 2;i <= n;++i)
41. **if**(dp[(1<<n)-1][i] + dis[i][0] < ans)
42. ans = dp[(1<<n)-1][i] +dis[i][0];
43. printf("%d\n",ans);
44. }
45. **return** 0;
46. }

[【HDU3001】Traveling](http://acm.hdu.edu.cn/showproblem.php?pid=3001)

【题目大意】10个点的TSP问题，但是要求每个点最多走两边，不是只可以走一次，所以要用三进制的状态压缩解决这个问题。可以预处理每个状态的第k位是什么。

【解析】和tsp问题相同,类似于上面那个题

【状态表示】【状态转移方程】同上题，具体见代码

【代码】

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/accry/article/details/6607703)

1. #include <cstdio>
2. #include <cstring>
3. #define INF 0x1f1f1f1f    //刚发现这里写0x1f1f1f跑的比0x1f1f1f1f差不多慢了一倍！Orz~
4. #define min(a,b) (a) < (b) ? (a) : (b)
5. **using** **namespace** std;
7. **int** N,M;
8. **int** tri[12] ={0,1,3,9,27,81,243,729,2187,6561,19683,59049};
9. **int** dig[59050][11];  //dig[state][k\_dig]  状态state的第k位是多少
10. **int** edge[11][11],dp[59050][11];
12. **int** main(){
13. **for**(**int** i = 0; i < 59050; ++i){
14. **int** t = i;
15. **for**(**int** j = 1; j <= 10; ++j){
16. dig[i][j] = t%3;
17. t /= 3;
18. **if**(t == 0)**break**;
19. }
20. }
21. **while**(scanf("%d%d",&N,&M) != EOF){
22. memset(edge,INF,**sizeof**(edge));
23. **int** a,b,c;
24. **while**(M --){
25. scanf("%d%d%d",&a,&b,&c);
26. **if**(c < edge[a][b])edge[a][b] = edge[b][a] = c;
27. }
28. memset(dp,INF,**sizeof**(dp));
29. **for**(**int** i = 1; i <= N; ++i)dp[tri[i]][i] = 0;
30. **int** ans = INF;
31. **for**(**int** S = 0; S < tri[N+1]; ++S){
32. **int** visit\_all = 1;
33. **for**(**int** i = 1; i <= N; ++i){
34. **if**(dig[S][i] == 0)visit\_all = 0;
35. **if**(dp[S][i] == INF)**continue**;
36. **for**(**int** j = 1; j <= N; ++j){
37. **if**(i == j)**continue**;
38. **if**(edge[i][j] == INF ||dig[S][j] >= 2)**continue**;
39. **int** newS = S + tri[j];
40. dp[newS][j] =min(dp[newS][j],dp[S][i] + edge[i][j]);
41. }
42. }
43. **if**(visit\_all){
44. **for**(**int** j = 1; j <= N; ++j)
45. ans = min(ans,dp[S][j]);
46. }
47. }
48. **if**(ans == INF){
49. puts("-1");
50. **continue**;
51. }
52. printf("%d\n",ans);
53. }
54. **return** 0;
55. }

[【POJ2288】Islands and Bridge](http://poj.org/problem?id=2288)

【题目大意】求汉密尔顿的一道变形问题，中间每个点有权值，关于最后得分的描述如下

Suppose there are n islands. The value of aHamilton path C1C2...Cn is calculated as the sum of three parts. Let Vi be thevalue for the island Ci. As the first part, we sum over all the Vi values foreach island in the path. For the second part, for each edge CiCi+1 in the path,we add the product Vi\*Vi+1. And for the third part, whenever three consecutiveislands CiCi+1Ci+2 in the path forms a triangle in the map, i.e. there is abridge between Ci and Ci+2, we add the product Vi\*Vi+1\*Vi+2.

这题要求让得分最高

【解析】发现每个点的状态由前面两个点确定，用DP（S,A,B）表示状态为S时，当前到达A，而上一个点是B时的最大得分，这个状态由DP（S',B,C）通过从B走到A得到，S'=S-(1<<A),即S'状态就是经过B和C但不经过A的一个状态，C是不同于A和B的一个点。

【状态转移】dp[S][A][B] =max(dp[S][A][B],dp[S'][B][C]+temp) 这里的temp指的是加上的得分即Vb\*Va+Va,如果构成三角关系（即A和C间有边），temp就要再加上Vb\*Va\*Vc.

【边界条件】DP((1<<A)+(1<<B),A,B)=Va+Vb+Va\*Vb(A和B间有边)表示

【代码】

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/accry/article/details/6607703)

1. #include <cstdio>
2. #include <cstring>
3. **using** **namespace** std;
4. **const** **int** MAXN = 13;
5. **const** **int** MAX\_S = 1<<(MAXN+1);
6. **long** **long** dp[MAX\_S][MAXN+1][MAXN+1];
7. **long** **long** way[MAX\_S][MAXN+1][MAXN+1];
8. **int** edge[MAXN+1][MAXN+1];
9. **long** **long** V[MAXN+1];
11. **int** N,M;
12. **int** main(){
13. **int** cas;
14. scanf("%d",&cas);
15. **while**(cas --){
16. memset(edge,0,**sizeof**(edge));
17. scanf("%d%d",&N,&M);
18. **for**(**int** i = 1; i <= N; ++i)
19. scanf("%d",&V[i]);
20. **if**(N == 1){
21. printf("%d 1\n",V[1]);
22. **continue**;
23. }
24. **int** a,b;
25. **while**(M --){
26. scanf("%d%d",&a,&b);
27. edge[a][b] = edge[b][a] = 1;
28. }
29. memset(dp,-1,**sizeof**(dp));
30. memset(way,0,**sizeof**(way));
31. **int** ii,jj;
32. **long** **long** temp;
33. **for**(**int** i = 1; i <= N; ++i)
34. **for**(**int** j = 1; j <= N; ++j){
35. **if**(i == j || !edge[i][j])**continue**;
36. ii = 1<<(i-1);
37. jj = 1<<(j-1);
38. temp = V[i]+V[j]+V[i]\*V[j];
39. dp[ii+jj][i][j] = temp;
40. way[ii+jj][i][j] = 1;
41. }
42. **for**(**int** S = 0; S < (1<<N); ++S)
43. **for**(**int** i = 1; i <= N; ++i){
44. **if**((S&(1<<(i-1))) == 0)**continue**;
45. **for**(**int** j = 1; j <= N; ++j){
46. **if**((S&(1<<(j-1))) ==0 || i == j || !edge[i][j])**continue**;
47. **for**(**int** k = 1; k <= N; ++k){
48. **if**(i == k || j == k ||(S&(1<<(k-1))) == 0)**continue**;
49. **int** newS = S -(1<<(i-1));
50. **if**(dp[newS][j][k] ==-1)**continue**;
51. **if**(!edge[j][k])**continue**;
52. temp =V[i]+V[i]\*V[j]+dp[newS][j][k];
53. **if**(edge[i][k])temp +=V[i]\*V[j]\*V[k];
54. **if**(dp[S][i][j] < temp){
55. dp[S][i][j] = temp;
56. way[S][i][j] =way[newS][j][k];  //如果dp更新，way直接更新
57. }
58. //如果dp不更新，但dp=temp，way加上原来的值
59. **else** **if**(temp ==dp[S][i][j])way[S][i][j] += way[newS][j][k];
60. }
61. }
62. }
63. **long** **long** ans = -1,num = 0;
64. **int** p = (1<<(N)) - 1;
65. **for** (**int** i = 1; i <= N; ++i)
66. **for** (**int** j = 1; j <= N; ++j){
67. **if**(i == j)**continue**;
68. **if** (ans < dp[p][i][j]){
69. ans = dp[p][i][j];
70. num = way[p][i][j];
71. }
72. **else** **if** (ans == dp[p][i][j])
73. num += way[p][i][j];
74. }
75. **if**(ans == -1){
76. puts("0 0");
77. **continue**;
78. }
79. printf("%lld %lld\n",ans,num/2);
80. }
81. **return** 0;
82. }

[【ZOJ4257】MostPowerful](http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemId=4257)

【题目大意】不超过10种气体，两两之间相互碰撞可以产生一定的能量，如a碰b，那么b气体就消失，自身不能碰自身，问最后所能得到的最大能量。

【题目解析】用10位二进制表示气体是否存在，0表示存在，1表示不存在，S（上一个状态）中的两种气体碰撞并且有一种消失，可以得到newS的状态（状态转移）

【状态表示】dp[state] 状态为state时的最大能量

【转移方程】dp[state] = max(dp[state],dp[state']+a[i][j])

【边界条件】dp[i] = 0;

【代码】

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/accry/article/details/6607703)

1. #include <cstdio>
2. #include <cstring>
3. **using** **namespace** std;
4. #define max(a,b) (a) > (b) ? (a) : (b)
6. **const** **int** MAXN = 10;
8. **const** **int** MAX\_S = 1 << 10;
10. **int** a[MAXN+1][MAXN+1];
12. **int** dp[MAX\_S];
14. **int** N;
16. **int** main(){
17. **while**(scanf("%d",&N) != EOF){
18. **if**(N == 0)**break**;
19. **for**(**int** i = 0; i < N; ++i)
20. **for**(**int** j = 0; j < N; ++j){
21. scanf("%d",&a[i][j]);
22. }
23. memset(dp,0,**sizeof**(dp));
24. **int** full = 1 << N;
25. **for**(**int** s = 0; s < full; ++s){
26. **for**(**int** i = 0; i < N; ++i){
27. **if**((s&(1<<i)))**continue**;
28. **for**(**int** j = 0; j < N; ++j){
29. **if**(i == j)**continue**;
30. **if**( (s&(1<<j)) )**continue**;
31. **int** newS = s + (1<<j);
32. dp[newS] = max(dp[newS],dp[s] + a[i][j]);
33. }
35. }
36. }
37. **int** ans = 0;
38. **for**(**int** s = 0; s < full ; ++s)
39. ans = max(ans,dp[s]);
40. printf("%d\n",ans);
41. }
42. **return** 0;
44. }

[【POJ2411】Mondriaan'sDream](http://poj.org/problem?id=2411)

【题目大意】一个矩阵，只能放1\*2的木块，问将这个矩阵完全覆盖的不同放法有多少种。

【解析】如果是横着的就定义11，如果竖着的定义为竖着的01，这样按行dp只需要考虑两件事儿，当前行&上一行，是不是全为1，不是说明竖着有空（不可能出现竖着的00），另一个要检查当前行里有没有横放的，但为奇数的1。

【状态表示】dp[state][i]第i行状态为state时候的方案数

【转移方程】dp[state][i] += dp[state'][i-1] state'为i-1行的，不与i行状态state冲突的状态

【边界条件】第一行 符合条件的状态记为1 即dp[state][0] = 1;

【代码】

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/accry/article/details/6607703)

1. #include <cstdio>
3. #include <cstring>
5. **using** **namespace** std;
7. **int** M,N;
9. **long** **long** dp[1<<11][11];
11. **bool** kexing[1<<11];
13. **int** full;
15. **inline** **bool** check(**int** in)
17. {
18. **int** bit=0;
19. **while**(in>0){
20. **if**( (in&1)==1)
21. bit++;
22. **else**{
23. **if**( (bit&1)==1)
24. **return** **false**;
25. bit=0;
26. }
27. in>>=1;
28. }
29. **if**( (bit&1)==1)
30. **return** **false**;
31. **return** **true**;
33. }
35. **inline** **bool** check2(**int** x1,**int** x2){
36. **if**( (x1|x2)!= full-1 )  //如果这里不用位运算，时间很长，要用约3000MS
37. **return** **false**;
38. **return** kexing[x1&x2];
40. }
42. **int** main(){
43. full = 1<<11;
44. **for**(**int** s = 0; s < full; ++s)**if**(check(s))kexing[s] = 1;
45. **while**(scanf("%d%d",&M,&N) != EOF){
46. **if**(N == 0 && M == 0)**break**;
47. memset(dp,0,**sizeof**(dp));
48. full = 1<<N;
49. **for**(**int** s = 0; s < full; ++s){
50. **if**(kexing[s])
51. dp[s][0] = 1;
52. }
53. **for**(**int** i = 1; i < M; ++i){
54. **for**(**int** s = 0; s < full; ++s){
55. **for**(**int** s1 = 0; s1 < full; ++s1){
56. **if**(!check2(s1,s))**continue**;
57. //if(dp[s1][i-1] == 0)continue;
59. /\*这一步判断的时间要大于位运算和+=的时间，如果先把这一步放在
61. check2前面，1300多MS，如果放在check2后面，610多MS，如果不加这
63. 一步，560MS，但如果check2用的不是位运算，将这一步加在check2前
65. 面3000MS左右（水过），如果不加这一步，TLE
67. \*/
68. dp[s][i] += dp[s1][i-1];
69. }
70. }
71. }
72. **int** S = (1<<N) - 1;
73. printf("%lld\n",dp[S][M-1]);
74. }
75. **return** 0;
77. }

[【HDU3681】PrisonBreak](http://acm.hdu.edu.cn/showproblem.php?pid=3681) 状态压缩DP+BFS+二分答案

2010杭州赛区的题目，以现在的水平遇到这种题也就能想一下，赛场上动手写这个题是不会的。前天做状压DP的时候又看到这个题，没想起来怎么做，昨天看了一下解题报告开始下手，遇到了各种问题。调试了N久，终于过了。

【题目大意】机器人从F出发，走到G可以充电，走到Y关掉开关，D不能走进，要求把所有开关关掉，且电量最少，并求出该最小电量。

【题目解析】机器人从出发点出发要求走过所有的Y，因为点很少，所以就能想到经典的TSP问题（起初我也想到了），但关于G点（不要YY）能充电的问题不知道怎么办，看了下解题报告才知道。G点可以充电，到达G点就把当前能量更新为电池容量然后继续走。因为每个G点只能充一次电，这就好像TSP中的每个点只能走一次一样（G和Y都可以走多次，但走到G充电后，该点就变为了S，而走到Y关上开关以后，Y也变成了S。这是一个很巧妙地想法，所以要求Y点只能关一次开关，G点只能充一次电，这就是TSP了。Orz赛场上可以秒杀这题的大神们），然后就是二分答案了，用状压DP判定当前电池容量的情况下是否能符合条件。

【状态表示】dp[s][i]表示到达当前i点状态为s时最大的剩余的能量

【转移方程】同TSP问题了

【边界条件】dp[1<<sid][sid] = rongliang.即出发点的能量就是电池容量

【代码】：

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/accry/article/details/6607703)

1. #include <cstdio>
3. #include <cstring>
5. #include <cmath>
7. #include <queue>
9. **using** **namespace** std;
11. #define INF 0x1f1f1f1f
13. **int** dp[32769][16];
15. **int** dist[16][16][16][16];
17. **int** di[4][2] = {{1,0},{-1,0},{0,1},{0,-1}};
19. **int** M,N,sid,nCnt,FinalState;
21. **char** map[16][16];
23. **struct** node{
24. **int** x,y;
25. node(){}
26. node(**int** \_x,**int** \_y):x(\_x),y(\_y){}
28. }nodes[16];

31. **inline** **void** BFS(node start)
33. {
34. queue<node> que;
35. **int** sx = start.x,sy = start.y;
36. dist[sx][sy][sx][sy] = 0;
37. que.push(start);
38. node cur;
39. **while**(!que.empty()){
40. cur = que.front();
41. que.pop();
42. **int** x = cur.x,y = cur.y,tx,ty;
43. **for**(**int** i = 0; i < 4; ++i){
44. tx = x + di[i][0];
45. ty = y + di[i][1];
46. **if**(tx < 0 || tx >= M || ty < 0 || ty >= N || map[tx][ty] == 'D')**continue**;
47. **if**(dist[sx][sy][tx][ty] == -1){
48. dist[sx][sy][tx][ty] = dist[sx][sy][x][y] + 1;
49. que.push(node(tx,ty));
50. }
51. }
52. }
54. }
56. **inline** **bool** ok(**int** s,**int** t){ //s状态中走过所有t状态中的城市
57. **if**(((s&t)&t) == t)**return** 1;
58. **return** 0;
59. }
60. **inline** **bool** check(**int** step){
61. **int** res = -1;
62. memset(dp,-1,**sizeof**(dp));
63. dp[1<<sid][sid] = step;
64. **int** full = 1<<nCnt;
65. **for**(**int** s = 0; s < full; ++s){
66. **for**(**int** i = 0; i < nCnt; ++i){
67. **if**((s&(1<<i)) == 0 || dp[s][i] == -1)**continue**;
68. **if**(ok(s,FinalState))res = max(res,dp[s][i]);
69. **for**(**int** j = 0; j < nCnt; ++j){
70. **int** temp = dist[nodes[i].x][nodes[i].y][nodes[j].x][nodes[j].y];
71. **if**(i == j || temp == -1 || (s&(1<<j)))**continue**;
72. temp = dp[s][i] - temp;
73. **if**(temp < 0)**continue**;
74. **int** newS = s + (1<<j);
75. dp[newS][j] = max(dp[newS][j],temp);
76. **if**(map[nodes[j].x][nodes[j].y] == 'G')dp[newS][j] = step;
77. }
78. }
79. }
80. **if**(res < 0)**return** 0;
81. **return** 1;
83. }

86. **inline** **int** solve(){
87. **int** low = 0,high = 300;
88. **int** mid;
89. **while**(low <= high){
90. mid = (low+high)/2;
91. **if**(check(mid))high = mid-1;
92. **else** low = mid+1;
93. }
94. **if**(low == 301)**return** -1;
95. **return** low;

98. }

101. **int** main(){
102. **while**(scanf("%d%d",&M,&N) != EOF){
103. **if**(M == 0 && N == 0)**break**;
104. nCnt = 0;
105. FinalState = 0;
106. **for**(**int** i = 0; i < M; ++i){
107. scanf(" %s",map[i]);
108. **for**(**int** j = 0; j < N; ++j){
109. **if**(map[i][j] == 'F'){
110. sid = nCnt;
111. nodes[nCnt++] = node(i,j);
112. FinalState += (1<<sid);
113. }
114. **else** **if**(map[i][j] == 'G')
115. nodes[nCnt++] = node(i,j);
116. **else** **if**(map[i][j] == 'Y'){
117. **int** tid = nCnt;
118. nodes[nCnt++] = node(i,j);
119. FinalState += (1<<tid);
120. }
121. }
122. }
123. memset(dist,-1,**sizeof**(dist));
124. **for**(**int** i = 0; i < nCnt; ++i)BFS(nodes[i]);
125. **int** ans = solve();
126. printf("%d\n",ans);
127. }
128. **return** 0;
130. }