题目要求：<27-P1123-取数游戏-题目要求.docx>

## DFS深搜算法

如果使用贪心法，每次去最大值，那么可以发现有问题。因为每取一个数，相邻的数就不能再取，那么每次决策就会影响下次结果，从而导致贪心决策不能最优化。

动态规划也是如此，在动态规划中，每一个分步决策不能影响后续结果，所以此算法也排除。

那么再观察数据范围：

对于100%的数据，N, M≤6,T≤20N,M≤6,T≤20。

这是一个较小的范围，所以可以考虑暴力枚举，而对于一个图，以及一个变化的状态，深度优先搜索（dfs） 应该是一个比较适合的算法。

再考虑每个数的状态。

显然，对于一个数，有取与不取两种状态，状态改变取决于周围取数的变化。我们用 mark[ i ][ j ] 记录点 ( i , j ) 附近有几个数，若 mark[ i ][ j ] ! = 0 就代表不能取这个数。

**这是一道比较简单的 dfs 题，主要需要确定算法和考虑标记状态，其他的就自然迎刃而解了。**

这一题每个数是否选择会影响后面的选择情况，所以需要用一个数组来保存 所以状态为当前选到那个数，之前选的数的和以及之前每个数是否选了 之后直接搜索即可。尽管复杂度较高，但因为存在大量的不合法情况所以可以通过 时间复杂度为O（2nm)

，但是，我们要注意的是我们如果只用1和0标记这个点是否走过，是不行的，因为当两个增加的点有重复部分的话，你return去掉一个那个重复部分就会被标记为0，但他旁边还是有一个选中，应当还是1，所以，我们计数每次++或--。

1. #include<bits/stdc++.h>//万能头文件
2. using namespace std;
3. const int d[8][2]={1,0,-1,0,0,1,0,-1,1,1,-1,1,1,-1,-1,-1};*//方向数组用来控制搜索时的方向*
4. int t,n,m,s[8][8],mark[8][8],ans,mx;
5. void dfs(int x,int y){*//搜索函数，表示搜索点(x,y)*
6. if(y==m+1){*//当y到边界时，搜索下一行*
7. dfs(x+1,1);
8. return;
9. }
10. if(x==n+1){*//当x到边界时，搜索结束，刷新最大值*
11. mx=max(ans,mx);
12. return;
13. }
15. dfs(x,y+1);*// 不取此数的情况*
17. if(mark[x][y]==0){ *//取此数的情况（需保证此数周围没有取其他数，即mark[i][j]==0）*
18. ans+=s[x][y];
19. for(int fx=0;fx<8;++fx){ *//标记周围的数*
20. ++mark[x+d[fx][0]][y+d[fx][1]];
21. }
22. dfs(x,y+1);
23. for(int fx=0;fx<8;++fx){ *//回溯*
24. --mark[x+d[fx][0]][y+d[fx][1]];
25. }
26. ans-=s[x][y];
27. }
29. }
30. int main(){
31. cin>>t;
32. while(t--){
33. memset(s,0,sizeof(s));
34. memset(mark,0,sizeof(mark));*//在做每个数据前都要初始化数组*
35. cin>>n>>m;
36. for(int i=1;i<=n;++i){
37. for(int j=1;j<=m;++j){
38. cin>>s[i][j];
39. }
40. }
41. mx=0;
42. dfs(1,1);*//从点(1,1)开始搜索*
43. printf("%d\n",mx);*//输出答案*
44. }
45. return 0;
46. }

优化：对于深搜来说，最常见的优化就是剪枝了，最容易看出的剪枝就是，搜完一个点之后，该点所在行之前的点就不用搜了，因为会和之前的点重复，于是在搜索的时候，每个点从遍历整个矩阵找下个点，改为找这个点所在行及其以下的所有点。再找新的剪枝，既然不用搜该点所在行以上的点，那么标记的时候就不用管上面的点，每次标记和删除标记都会少三次++或—

## 状压DP：

设f[i][j],表示现在枚举到第i行，用一个二进制数表示这行的状态，如果某一位是1，说明这行这个位置会被取，因为这行的状态与上一行有关，所以f[i][j]=max(f[i][j],f[i-1][k]+sum)。(sum表示这行取的数的总和),我们可以采用预处理每一行可用的状态来优化时间。

1. #include<bits/stdc++.h>
2. const int maxn=7;
3. const int maxm=1<<maxn;
4. using namespace std;
5. int T,n,m,f[maxn][maxm],a[maxn][maxn];
6. vector<int>t;
7. void work(){
8. for(int i=0;i<(1<<m);i++)if(!(i&(i>>1))&&!(i&(i<<1)))t.push\_back(i);
9. }
10. int main(){
11. scanf("%d",&T);
12. for(;T;T--){
13. int ans=0;
14. memset(f,0,sizeof(f));
15. scanf("%d %d",&n,&m);
16. for(int i=1;i<=n;i++)for(int j=1;j<=m;j++)scanf("%d",&a[i][j]);
17. work();
18. for(int i=1;i<=n;i++){
19. for(int j=0;j<t.size();j++)
20. for(int k=0;k<t.size();k++){
21. int u=t[j],v=t[k];
22. if((u&v)||(u&(v>>1))||(u&(v<<1)))continue;
23. int num1=0;
24. for(int l=0;l<m;l++)if((u&(1<<l)))num1+=a[i][m-l];
25. f[i][u]=max(f[i][u],f[i-1][v]+num1);
26. }
27. }
28. for(int i=0;i<t.size();i++)ans=max(ans,f[n][t[i]]);
29. printf("%d\n",ans);
30. t.clear();
31. }
32. return 0;
33. }

状压附解：

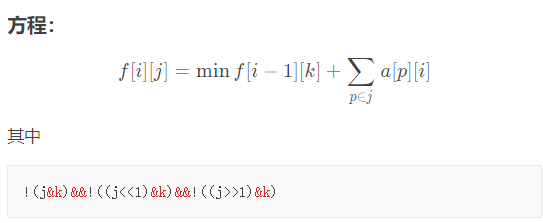
数据范围如此之小，一方面想到用爆搜做，另一方面，看到熟悉的约束条件——

每一个位置只会影响到8联通的格子。就可以发现，每一列的取值只会影响到左右两列，每一行的取值只会影响到上下两行。

所以，我的思路是：

设计状态：f[i][j]为考虑前i列，最后一列的摆放形如j的最大取值。

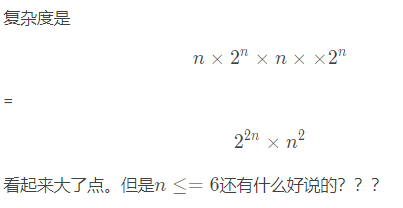
这样，第i行之后的取值和i行前的取值就一点关系都没有了。



表示八个方向不影响。

目前为止都很顺利。但是数字之间还不能上下相邻啊，而我们枚举状态要判重似乎冗余了点。

所以我们要用另一个数组fun[]来存每一行基本合法（指本身不冲突）的状态；这样我们ff中的第二维就可以简单地表示成该状态在fun数组中的下标了。



注意事项：因为我们的状态中有一维对于统计答案是不足取的，所以最后我们还要枚举f[m]中的每一个状态的最优值取max。

附上文件路径：[**状态压缩动态规划**](状态压缩动态规划.docx)

## 我的代码：

附上文件链接：<27-P1123-取数游戏.cpp>

1. #include<stdio.h>
2. #include<math.h>
3. typedef struct{
4. int n,m;
5. int a[10][10];
6. }list;
7. list num[21];
8. int t,check[10][10],ans,res;
9. *//check数据用来记录该点附近有几个数*
10. int dirx[]={0,1,1, 1, 0,-1,-1,-1};
11. int diry[]={1,1,0,-1,-1,-1, 0, 1};
12. void dfs(int x,int y,int i);
13. int main(void)
14. {
15. scanf("%d",&t);*//读入数据的组数*
16. for(int i=0;i<t;i++)
17. {
18. scanf("%d%d",&num[i].n,&num[i].m);
19. *//读入当前组数据的行数和列数*
20. for(int j=1;j<=num[i].n;j++)
21. for(int k=1;k<=num[i].m;k++)
22. scanf("%d",&num[i].a[j][k]);
23. *//读入数据*
24. }
25. for(int i=0;i<t;i++)
26. {
27. ans=0;
28. dfs(1,1,i);
29. printf("%d\n",ans);
30. }
31. }
32. void dfs(int x,int y,int i)
33. {
34. if(y==num[i].m+1) {dfs(x+1,1,i);return;}*//如果遍历到行尾则行数加一后再次搜索*
35. if(x==num[i].n+1) {ans=((ans>res)?ans:res);return;}
36. *//如果遍历到列尾则比较当前得到的结果与已得到的上一轮结果，取最大值*
37. dfs(x,y+1,i);*//若均不是以上两种情况则同行向右搜索，也即不取此数*
38. if(check[x][y]==0)
39. {*//如果该点未被标记，即周围没有取其他数，那么就加上这一点的值*
40. res+=num[i].a[x][y];
41. for(int p=0;p<8;p++)*//将该点周围的值进行标记*
42. check[x+dirx[p]][y+diry[p]]++;
43. dfs(x,y+1,i);*// 向下方搜索*
44. for(int p=0;p<8;p++)*//回溯*
45. check[x+dirx[p]][y+diry[p]]--;
46. res-=num[i].a[x][y];
47. }
48. }