对抗搜索：  
一、 对抗搜索的适用范围  
在博弈论题目中，如果决策双方的获胜条件是截然相反的，即一方要求得分越高越好，另一方要求得分越低越好，这时我们就可以用上对抗搜索算法。  
  
二、对抗搜索的主要思想  
对抗搜索的核心思想就是dfs dfsdfs遍历一遍博弈树。  
  
不难想到，如果博弈树非常庞大，在不加优化的情况下，对抗搜索的时间效率是十分低下的。  
  
因此，我们就需要对对抗搜索进行一定的优化。  
三、对抗搜索的优化  
对抗搜索的优化一般来讲有两种：记忆化和 Alpha−Beta Alpha-BetaAlpha−Beta剪枝 。

**源代码：**

**#include <iostream>**

**#include <string.h>**

**#include <stdlib.h>**

**#include <stdio.h>**

**using namespace std;**

**#define MAX 0x7FFFFFF**

**const int link[19] =**

**{0,**

**17,33, 8387, 18, 4371, 4500,**

**36,8518, 8776, 21, 4758, 4887,**

**5016,5145, 41, 69, 71, 73};**

**int tri[10];**

**#define TRI\_MASK 15**

**#define DIRECTION\_MASK 7**

**#define TRI\_BIT 4**

**#define DIRECTION\_BIT 3**

**bool used[19];**

**int addStick(int pos)**

**{**

**int score;**

**int l;**

**int p;**

**int d;**

**score = 0;**

**used[pos] = 1;**

**l = link[pos];**

**p = l & TRI\_MASK;**

**l = l >> TRI\_BIT;**

**d = l & DIRECTION\_MASK;**

**tri[p] |= d;**

**if(tri[p]== 7)**

**{**

**score++;**

**}**

**l = l >> DIRECTION\_BIT;**

**p = l & TRI\_MASK;**

**if(p != 0)**

**{**

**l = l >> TRI\_BIT;**

**d = l & DIRECTION\_MASK;**

**tri[p] |= d;**

**if(tri[p]== 7)**

**{**

**score++;**

**}**

**}**

**returnscore;**

**}**

**void removeStick(int pos)**

**{**

**int l;**

**int p;**

**int d;**

**used[pos] = 0;**

**l = link[pos];**

**p = l & TRI\_MASK;**

**l = l >> TRI\_BIT;**

**d = l & DIRECTION\_MASK;**

**tri[p] &= ~d;**

**l = l >> DIRECTION\_BIT;**

**p = l & TRI\_MASK;**

**if(p != 0)**

**{**

**l = l >> TRI\_BIT;**

**d = l & DIRECTION\_MASK;**

**tri[p] &= ~d;**

**}**

**}**

**int originstep;**

**#define LOWEST 5**

**#define HIGHEST 4**

**int MinValue(int step, int baseValue, intalpha, int beta);**

**int MaxValue(int step, int baseValue, intalpha, int beta)**

**{**

**int i;**

**int max;**

**int n, t;**

**if(step ==18)**

**return0;**

**/\*if(step ==17)//加上这个WA，难道是避开了alpha、beta的缘故？**

**return 1;\*/**

**max = -MAX;**

**for(i = 1;i <= 18 ; i ++)**

**{**

**if(used[i]== 1)**

**{**

**continue;**

**}**

**n = addStick(i);**

**if(n> 0){**

**t = n + MaxValue(step + 1, n +baseValue, alpha, beta);**

**}else{**

**t = MinValue(step + 1, baseValue,alpha, beta);**

**}**

**removeStick(i);**

**if (max< t )**

**max = t;**

**if(max+ baseValue >= beta)**

**{**

**returnmax;**

**}**

**if(alpha< max + baseValue)**

**{**

**alpha = max + baseValue;**

**}**

**if(alpha> HIGHEST){**

**returnmax;**

**}**

**}**

**return max;**

**}**

**int MinValue(int step, int baseValue, intalpha, int beta)**

**{**

**int i;**

**int min;**

**int n, t;**

**if(step ==18) //|| step == 17)**

**return0;**

**min = MAX;**

**for(i = 1;i <= 18 ; i ++)**

**{**

**if(used[i]== 1)**

**{**

**continue;**

**}**

**n = addStick(i);**

**if(n> 0){**

**t = MinValue(step + 1, baseValue,alpha, beta);**

**}else{**

**t = MaxValue(step + 1, baseValue,alpha, beta);**

**}**

**removeStick(i);**

**if (min> t)**

**min = t;**

**if(min+ baseValue <= alpha)**

**{**

**returnmin;**

**}**

**if(beta> min + baseValue)**

**{**

**beta = min + baseValue;**

**}**

**if(beta< LOWEST){**

**returnmin;**

**}**

**}**

**return min;**

**}**

**#define LEFT 0**

**#define RIGHT 1**

**#define MIDDLE 2**

**const int dot[19][2] ={**

**{0,0},**

**{1, 2}, {1, 3},**

**{2,3},**

**{2,4}, {2, 5}, {3, 5}, {3, 6},**

**{4,5}, {5, 6},**

**{4,7}, {4, 8}, {5, 8}, {5, 9}, {6, 9}, {6, 10},**

**{7,8}, {8, 9}, {9, 10}};**

**int calLink()**

**{**

**freopen( "t.in","w", stdout );**

**int i;**

**int j,index;**

**char c;**

**int result;**

**int pos;**

**while(1)**

**{**

**cin >> j;**

**result = 0;**

**for(i =0; i < j; i++){**

**cin>> index >> c;**

**if(c== 'l'){**

**pos = LEFT;**

**}elseif(c == 'r'){**

**pos = RIGHT;**

**}else{**

**pos = MIDDLE;**

**}**

**pos = 1 << pos;**

**result = result << 7;**

**result |= (pos << 4) |index;**

**}**

**cout << result << ", ";**

**}**

**return 1;**

**}**

**int main()**

**{**

**int cases;**

**boolmaxTurn;**

**intoriginValue;**

**int i, j,k;**

**int from,to;**

**int n;**

**int losed;;**

**scanf("%d",&cases);**

**for(k = 1;k <= cases; k++){**

**memset(tri, 0, sizeof(tri));**

**memset(used, 0, sizeof(used));**

**maxTurn = 1;**

**originValue = 0;**

**losed = 0;**

**scanf("%d",&originstep);**

**for(i =1; i <= originstep; i++){**

**scanf("%d%d",&from, &to);**

**for(j= 1; j <= 18; j++){**

**if(dot[j][0]== from && dot[j][1] == to)**

**{**

**break;**

**}**

**}**

**n = addStick(j);**

**if(n== 0){**

**maxTurn = !maxTurn;**

**}else{**

**if(maxTurn){**

**originValue += n;**

**}else{**

**losed += n;**

**}**

**}**

**}**

**printf("Game%d: ", k);**

**if(losed<= 4 && originValue <= 4){**

**if(maxTurn){**

**originValue +=MaxValue(originstep, originValue, -MAX, MAX);**

**}else{**

**originValue +=MinValue(originstep, originValue, -MAX, MAX);**

**}**

**if(originValue> HIGHEST){**

**printf("A");**

**}elseif(originValue < LOWEST){**

**printf("B");**

**}else{**

**int\*a = (int \*)0;**

**\*a = 0;**

**}**

**}else if(originValue > 4){**

**printf("A");**

**}else{**

**printf("B");**

**}**

**printf("wins.\n");**

**}**

**return 1;**

**}**