****



**程序设计综合实践课程报告**

**搜索1实验**

**学生姓名**

**学院名称 智能与计算学部**

**专 业 工科试验班(智能与计算类)**

**学 号**

# 1. 正方形

## 1.1题目分析

**用所有的木棒可以求出正方形的边长，然后凑四个正方形边长即可。**

## 1.2 题目代码（带注释）

|  |
| --- |
| #include<iostream>  #include <algorithm>  #include<cstring>  using namespace std;  int N, n, a[21], sum, ave, v[21], flag;//a数组用来保存木棍长度，v数组用来检测是否使用过该边  int dfs(int t, int summ, int gs, int av)  {      if (summ == av)      {          gs++;          summ = 0;          t = 1;          if (gs == 4)          {              flag = 1;              return 1;          }      }      for (int i = t; i < n; i++)      {          if ((v[i] == 0) && (summ + a[i] <= av))          {              v[i]=1;              if(dfs(i+1,summ+a[i],gs,av))                  return 1;              v[i] = 0;          }      }      return 0;  }  int main()  {      cin >> N;      while (N--)      {          cin >> n;          memset(a, 0, sizeof(a));          memset(v, 0, sizeof(v));          sum = 0;          flag = 0;          for (int i = 0; i < n; i++)          {              cin >> a[i];              sum += a[i];          }          if (sum % 4 != 0)          {              cout << "no" << endl;          }          else          {              ave = sum / 4;              dfs(0, 0, 0, ave);              if (flag == 1)              {                  cout << "yes" << endl;              }              else                  cout << "no" << endl;          }      }        return 0;  } |
|  |

# 2. prime circle

## 2.1题目分析

**用两个数组，一个数组判断是否使用过这个数，一个数组存储这个圈。在找不到与前面的数相加为素数并没有使用过时，用i--，再看前面的数。**

## 2.2 题目代码（带注释）

|  |
| --- |
| #include <iostream>  #include<cstdint>  #include<cstdio>  #include<algorithm>  #include<cmath>  #include<cstring>  using namespace std;  static const int MAX\_N = 1e6 + 5;  typedef long long ll;  int n;  int vv[25];  bool vis[25];  bool is\_prime(int x) {      if (x == 1) return false;      if (x == 2) return true;      if (x % 2 == 0) return false;      for (int i = 3; i \* i <= x; i += 2) {          if (x % i == 0) return false;      }      return true;  }    void dfs(int s, int len) {      if (len == n && is\_prime(vv[len] + vv[1])) {          for (int i = 1; i < len; i++) {              printf("%d ", vv[i]);          }          printf("%d\n", vv[len]);          return;      }      if (len > n) return;      for (int i = 1; i <= n; i++) {          if (vis[i]) continue;          if (is\_prime(s + i)) {              vv[len + 1] = i;              vis[i] = true;              dfs(i, len + 1);              vis[i] = false;          }      }  }  int main() {      int cas = 1;      while (scanf("%d", &n) != EOF) {          memset(vis, false, sizeof(vis));          printf("Case %d:\n", cas++);          vis[1] = true;          vv[1] = 1;          dfs(1, 1);          printf("\n");      }      return 0;  } |
|  |

# 3. 棋盘问题

## 3.1题目分析

**可以利用dfs解决棋盘类的问题。**

## 3.2 题目代码（带注释）

|  |
| --- |
| #include<bits/stdc++.h>  using namespace std;  char a[9][9];  bool b[100];  int n, k, ans;  void dfs(int x, int y)//x为行数，y为落子数  {      if (y == k)      {          ans++;          return;      }      for (int i = x; i <= n; i++)//从x行开始搜，x为上一行数+1          for (int j = 1; j <= n; j++)//搜索当前行的每一列          {              if (a[i][j] == '#' && b[j] == true)              {                  b[j] = false;                  dfs(i + 1, y + 1);                }          }      return;  }  int main()  {      while (cin >> n >> k)      {          if (n == -1 && k == -1) return 0;          memset(b, true, sizeof(b));          for (int i = 1; i <= n; i++)              for (int j = 1; j <= n; j++)                  cin >> a[i][j];          ans = 0;          dfs(1, 0);          cout << ans << endl;      }      return 0;  } |
|  |

# 4. 非常可乐

## 4.1题目分析

**利用循环和与运算可以解决。**

## 4.2 题目代码（带注释）

|  |
| --- |
| #include<cstdio>  #include<iostream>  using namespace std;  int gcd(int a, int b)  {      return b ? gcd(b, a % b) : a;  }  int main()  {      int a, b, c;      while (scanf("%d%d%d", &a, &b, &c), a + b + c)      {          a /= gcd(b, c);          if (a & 1)printf("NO\n");          else printf("%d\n", a - 1);      }      return 0;  } |
|  |

# 5. 迷宫问题

## 5.1题目分析

**自己重新写了一些关于栈的函数，利用深度优先搜素遍历解决该问题。**

## 5.2 题目代码（带注释）

|  |
| --- |
| #include<stdio.h>  #include<stdlib.h>    //迷宫问题    struct path  {      int x;            //横座标      int y;      //纵座标      struct path\* next;  };    typedef struct Link  {      struct path\* head;  }link;      //模拟为栈  void push(link\* que, int x, int y)  {      struct path\* p;      p = (struct path\*)malloc(sizeof(struct path));      p->x = x;      p->y = y;      p->next = que->head;;      que->head = p;      /\*       que->rear->next = p;       que->rear = p;       que->rear->next = NULL;      \*/  }    void pop(link\* que)  {      //如果不合适将刚入栈的节点删除      que->head = que->head->next;  }    /\*  void pop2(link \*que, int \*x, int \*y)  {   struct path \*temp;   temp = que->head->next;   que->head->next = temp->next;      //将temp结点出队   \*x = temp->x;   \*y = temp->y;   free(temp);  }  \*/    void r(int a[5][5], int x, int y, link\* que)  {      if (x < 5 && x >= 0 && y < 5 && y >= 0)      {          if (x == 4 && y == 4)              return;          if (a[x][y + 1] == 0 && (y + 1) < 5)          {              y++;              push(que, x, y);              r(a, x, y, que);              return;          }          if (a[x + 1][y] == 0 && (x + 1) < 5)          {              x++;              push(que, x, y);              r(a, x, y, que);              return;          }          if (a[x - 1][y] == 0 && x - 1 >= 0)          {              a[x][y] = 1;              pop(que);              x--;              r(a, x, y, que);              return;          }          if (a[x][y - 1] == 0 && y - 1 >= 0)          {              a[x][y] = 1;              pop(que);              y--;              r(a, x, y, que);              return;          }      }  }    int main(void)  {      int a[5][5];      int i, j;      int x, y;      for (i = 0; i < 5; i++)      {          for (j = 0; j < 5; j++)          {              scanf("%d", &a[i][j]);          }      }      link\* top, \* temp;      top = temp = (link\*)malloc(sizeof(link));      top->head = (struct path\*)malloc(sizeof(struct path));      r(a, 0, 0, top);      printf("(0, 0)\n");      //此处数组是为了存储路径，因为使用的是栈结构，所以存储的路径必然是倒序！      int PATH[25] = { 0 };      int count = 0;        //将栈倒序的路径存储进数组中，二维数组太占用空间所以此处我们使用一维数组，并使用数学方法存储！！      while (top->head->next != NULL)      {          PATH[count] = top->head->x \* 10 + top->head->y;          ++count;          top->head = top->head->next;      }      for (i = count - 1; i >= 0; --i)      {          int num\_x = PATH[i] / 10;  //从数据中提取出X          int num\_y = PATH[i] % 10; //从数组中提取出Y          //打印路径          printf("(%d, %d)\n", num\_x, num\_y);      }      return 0;  } |
|  |

# 6. Catch That Cow

## 6.1题目分析

**农夫有三种方式进行移动，每种方式需要的时间相同，都是一分钟  
1.向前移动一米（+1）  
2.向后移动一米（-1）  
3.移动到2\*n（移动到二倍农夫当前米数）  
牛不动  
求农夫走到牛所在位置的最短时间（分钟）并输出。**

## 6.2 题目代码（带注释）

|  |
| --- |
| #include <stdio.h>  #include <string.h>  #include <queue>  using namespace std;    const int N = 1000000;  int map[N + 10];  int n, k;  struct node  {      int x, step;  };    int check(int x)  {      if (x < 0 || x >= N || map[x])          return 0;      return 1;  }    int bfs(int x)  {      int i;      queue<node> Q;      node a, next;      a.x = x;      a.step = 0;      map[x] = 1;      Q.push(a);      while (!Q.empty())      {          a = Q.front();          Q.pop();          if (a.x == k)              return a.step;          next = a;          //每次都将三种状况加入队列之中          next.x = a.x + 1;          if (check(next.x))          {              next.step = a.step + 1;              map[next.x] = 1;              Q.push(next);          }          next.x = a.x - 1;          if (check(next.x))          {              next.step = a.step + 1;              map[next.x] = 1;              Q.push(next);          }          next.x = a.x \* 2;          if (check(next.x))          {              next.step = a.step + 1;              map[next.x] = 1;              Q.push(next);          }      }      return -1;  }    int main()  {      int ans;      while (~scanf("%d%d", &n, &k))      {          memset(map, 0, sizeof(map));          ans = bfs(n);          printf("%d\n", ans);      }      return 0;  } |
|  |

# 7. Find a way

## 7.1题目分析

**以两个人为基准点先后BFS，每次找到KFC的位置后，在新开的一个数组(ans)中对应的地方记录路程，最后遍历ans中最小的便是最短路程。**

## 7.2 题目代码（带注释）

|  |
| --- |
| #include<stdio.h>  #include<string.h>  #include<queue>  using namespace std;  const int maxn = 202;  const int INF = 0x3f3f3f3f;  char map[maxn][maxn], vis[maxn][maxn];  int dis[maxn][maxn][2];//记录下到每个@所走的步数，其中第三维代表有两个人走  int n, m, flag;  struct node  {      int x, y, step;  };  bool judge(int x, int y)  {      if (x >= 0 && x < n && y >= 0 && y < m && !vis[x][y] && map[x][y] != '#')          return 1;      return 0;  }  int to[][2] = { 0,1,0,-1,1,0,-1,0 };  void bfs(int x, int y)  {      node now, next;      queue<node>q;      now.x = x, now.y = y, now.step = 0;      q.push(now);      vis[x][y] = 1;      while (!q.empty())      {          now = q.front();          if (map[now.x][now.y] == '@')//遇到@就记录下走了多少步              dis[now.x][now.y][flag] = now.step;          for (int i = 0; i < 4; i++)          {              int xx = now.x + to[i][0], yy = now.y + to[i][1];              if (judge(xx, yy))              {                  vis[xx][yy] = 1;                  next.x = xx, next.y = yy, next.step = now.step + 1;                  q.push(next);              }          }          q.pop();      }  }  int main()  {      while (~scanf("%d%d", &n, &m))      {          int i, j, s = INF;          for (i = 0; i < n; i++)              for (j = 0; j < m; j++)                  dis[i][j][0] = dis[i][j][1] = INF;//有可能有的@根本走不通，所以要初始化无穷大          for (i = 0; i < n; i++)              scanf("%s", map[i]);          for (i = 0; i < n; i++)              for (j = 0; j < m; j++)              {                  if (map[i][j] == 'Y')                  {                      flag = 0;                      memset(vis, 0, sizeof(vis));                      bfs(i, j);                  }                  if (map[i][j] == 'M')                  {                      flag = 1;                      memset(vis, 0, sizeof(vis));                      bfs(i, j);                  }              }          for (i = 0; i < n; i++)              for (j = 0; j < m; j++)              {                  if (map[i][j] == '@' && s > dis[i][j][0] + dis[i][j][1])//找到最小步数和                      s = dis[i][j][0] + dis[i][j][1];              }          printf("%d\n", s \* 11);      }      return 0;  } |
|  |