**P1031**

解题思路

首先是预处理，用yc[i][j]来存储 第i个单词 后连接 第j个单词 的 最小重叠部分（mt函数）

后来预处理完了之后就是深搜:

先从第一个到最后一个单词看一看哪个单词是指定字母为开头的，作为深搜的第一个单词，同时标记使用过一次（vis[i]++)

然后继续搜即可

#include <cstdio>

#include <iostream>

#include <string>

#include <cmath>

using namespace std;

int n; /\* 单词数 \*/

string tr[30]; /\* 存储字符串 \*/

int yc[30][30]; /\* 两个字母的最小重叠部分 \*/

int vis[30]; /\* 判断单词使用频率. \*/

int mt( int x, int y ) /\* mt函数，返回x单词后连接一个y单词的最小重叠部分 \*/

{

bool pp = true;

int ky = 0;

for ( int k = tr[x].size() - 1; k >= 0; k-- ) /\* 从x单词尾部向前看看最小重叠部分是从哪里开始的，以为因为是倒着来，所以保证是最小的 \*/

{

for ( int kx = k; kx < tr[x].size(); kx++ )

{

if ( tr[x][kx] != tr[y][ky++] )

{

pp = false;

break;

}

}

if ( pp == true ) /\* 如果说当前以k为开头的前一个单词后缀 ,是后面单词的前缀，就马上返回重叠部分。（tr[x].size()-k是找出来的规律） \*/

{

return(tr[x].size() - k);

}

ky = 0;

pp = true; /\*不行就继续 \*/

}

return(0);

} /\* 可能这里有点难理解。可以手动模拟一下 \*/

char ch; /\* 开头字母 \*/

int ans = -1; /\* 答案 \*/

int an = 0; /\* 每次搜到的当前最长串 \*/

void dfs( int p ) /\* p为尾部单词编号(p的后缀就是“龙”的后缀，因为p已经连接到”龙“后面了) \*/

{

bool jx = false;

for ( int j = 1; j <= n; j++ )

{

if ( vis[j] >= 2 )

continue; /\* 使用了两次就跳过 \*/

if ( yc[p][j] == 0 )

continue; /\* 两单词之间没有重合部分就跳过 \*/

if ( yc[p][j] == tr[p].size() || yc[p][j] == tr[j].size() )

continue; /\* 两者存在包含关系就跳过 \*/

an += tr[j].size() - yc[p][j]; /\* 两单词合并再减去最小重合部分 \*/

vis[j]++; /\* 使用了一次 \*/

jx = true; /\* 标记一下当前已经成功匹配到一个可以连接的部分 \*/

dfs( j ); /\* 接上去 \*/

an -= tr[j].size() - yc[p][j]; /\* 回溯，就要再减回去那一部分长度 \*/

vis[j]--; /\* 回溯,使用-- \*/

}

if ( jx == false ) /\* jx==false说明不能再找到任何一个单词可以相连了 \*/

{

ans = max( ans, an ); /\* 更新ans \*/

}

return;

}

int main()

{

scanf( "%d", &n );

for ( int i = 1; i <= n; i++ )

cin >> tr[i];

cin >> ch;

for ( int i = 1; i <= n; i++ )

{

for ( int j = 1; j <= n; j++ )

{

yc[i][j] = mt( i, j );

}

} /\* 预处理yc数组。yc[i][j]就表示，i单词后连接一个j单词的最小重叠部分 \*/

/\*

\* 比如 i表示at,j表示att. yc[i][j]就为2 但是yc[j][i]就为0.

\* 预处理是一个关键

\*/

for ( int i = 1; i <= n; i++ ) /\* 从头到尾看一下有没有以指定开头字母为开头的单词 \*/

{

if ( tr[i][0] == ch ) /\* 如果有，就以当前单词为基准进行搜索。 \*/

{

vis[i]++; /\* 使用过一次 \*/

an = tr[i].size(); /\* 更新当前串长度 \*/

dfs( i ); /\* 接上 \*/

vis[i] = 0; /\* 消除影响 \*/

}

}

printf( "%d", ans );

return(0);

}

**P1050**

一想到去重，那么我们就可以把元素丢到一个集合里。

从此不用担心重复问题。

基本思路：

创建一个查找集合，一个去重答案集合

（由于集合的插入和查找的时间复杂度皆为O(log n)，加上这题n的数据范围只有100，所以可以使用集合）

枚举。

时间复杂度 O（n^2longn）

#include <cstdio>

#include <iostream>

#include <set>

using namespace std;

set<int> num; /\* 查找用集合 \*/

set<int> res; /\* 去重用集合 \*/

int n, a[105], tmp;

int main()

{

cin >> n;

for ( int i = 1; i <= n; ++i )

{

cin >> a[i]; /\* 输入数值 \*/

num.insert( a[i] ); /\* 用集合方便查找 \*/

}

for ( int i = 1; i <= n; ++i ) /\* 枚举a[i] \*/

{

for ( int j = i + 1; j <= n; ++j ) /\* 枚举a[j] \*/

{

int tmp = a[i] + a[j]; /\* 接下来定义tmp，它是所枚举的两数之和 \*/

set<int>::iterator it = num.find( tmp ); /\* 直接在集合中查找tmp \*/

/\* 注意，find函数返回的是一个迭代器，所以要用一个迭代器来存储find的返回值 \*/

if ( it != num.end() ) /\* 在find中，如果没有找到集合中的元素，那么迭代器会在集合的末尾。 \*/

{ /\* 所以如果it不在集合末尾，那么说明it在集合中出现过 \*/

res.insert( \*it ); /\* 在去重集合中插入 \*/

}

}

}

cout << res.size(); /\* 直接输出集合内元素数量 \*/

return(0);

}

**P1774**

一个找最短路的问题，可以用深搜来写，注意是一个无向图。

#include <stdio.h>

#include <string.h>

int min, book[101], n, e[101][101], x, y;

void dfs( int cur, int dis )

{

int j;

if ( dis > min )

return;

if ( cur == y )

{

if ( dis < min )

min = dis;

return;

}

for ( j = 1; j <= n; j++ )

{

if ( e[cur][j] != 99999999 && book[j] == 0 )

{

book[j] = 1;

dfs( j, dis + e[cur][j] );

book[j] = 0;

}

}

return;

}

int main( void )

{

int i, j, m, a, b, c;

while ( ~scanf( "%d %d", &n, &m ) )

{

memset( book, 0, sizeof(book) );

for ( i = 1; i <= n; i++ )

for ( j = 1; j <= n; j++ )

if ( i == j )

e[i][j] = 0;

else

e[i][j] = 99999999;

for ( i = 1; i <= m; i++ )

{

scanf( "%d %d %d", &a, &b, &c );

e[a][b] = e[b][a] = c;

}

scanf( "%d %d", &x, &y );

min = 99999999;

book[x] = 1;

dfs( x, 0 );

if ( min == 99999999 )

printf( "No path\n" );

else

printf( "%d\n", min );

}

return(0);

}

**P1046**

详细解释思路（当苹果数大于等于盘子数时的分法）：一种是目前所有的盘子都放一个苹果，盘子数不变，即：apple(m-n,n);

另一种是将一个盘子不放，再来进行思考，即：apple(m,n-1);

两种情况的总和就是答案；

接下来就用一个例子来详细解释：

现在有5个苹果和3个盘子；

因为苹果数大于盘子数，所以分成两种：

第一种：目前所有盘子放一个苹果（目前盘子数3个）apple(2,3);

第二种：拿一个盘子不放苹果，剩下的盘子继续考虑，apple(5,2);

apple(2,3):因为苹果数小于盘子数，所以apple(2,2);

apple(2,2):因为苹果数等于盘子数，所以又分为两种：

第一种：目前所有盘子放一个苹果（目前盘子数2个）apple(0,2); 没有苹果，达到边界，返回值1；

第二种：拿一个盘子不放苹果，剩下的盘子继续考虑，apple(2,1); 盘子数只有1个，达到边界，返回值1；

apple(5,2):因为苹果数大于盘子数，所以又分为两种：

第一种：目前所有盘子放一个苹果（目前盘子数2个)apple(3,2);

第二种：拿一个盘子不放苹果，剩下的盘子继续考虑，apple(5,1); 因为盘子数只有1个，达到边界，返回值1；

apple(3,2):因为苹果数大于盘子数，所以又分为两种：

第一种：目前所有盘子放一个苹果（目前盘子数2个)apple(1,2); 苹果数只有1个，达到边界，返回值1；

第二种：拿一个盘子不放苹果，剩下的盘子继续考虑，apple(3,1); 盘子数只有1个，达到边界，返回值1；

所有的值相加得到最后的结果5，记录在数组sum中，最后输出；

#include <iostream>

#include <cstdio>

#include <cstdlib>

#include <cmath>

using namespace std;

int sum[10000]; /\* 用来记录每一组数据最后的结果 \*/

int apple( int m, int n ) /\* m是苹果数，n是盘子数 \*/

{

if ( m == 0 || m == 1 || n == 1 ) /\* 边界：当苹果只有一个或者没有苹果时或者盘子只有一个时，只有一种放法，所以达到边界，返回值； \*/

return(1);

if ( m < n ) /\* 当苹果数少于盘子数，就只有m个盘子作用，所以接下来就计算m个苹果和m个盘子； \*/

return(apple( m, m ) );

if ( m >= n ) /\* 如果苹果数大于等于盘子数，分成两个部分，一种是目前所有的盘子都放一个苹果，另一种是拿一个盘子不放； \*/

return(apple( m - n, n ) + apple( m, n - 1 ) );

}

int main()

{

int n, m, s; /\* 其中s是数据的个数； \*/

cin >> s;

for ( int i = 1; i <= s; i++ )

{

cin >> m >> n; /\* m是苹果数，n是盘子数 \*/

sum[i] = apple( m, n ); /\* 将每一组的苹果和盘子的到的结果记录在sum数组之中； \*/

}

for ( int j = 1; j <= s; j++ )

cout << sum[j] << endl;

return(0);

}

**P1601**

#include <algorithm>

#include <iostream>

using namespace std;

int h[] = { 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900, 2674440, 9694845, 35357670, 129644790, 477638700, 1767263190 };

int nIndex[] = { 1, 2, 4, 9, 23, 65, 197, 626, 2056, 6918, 23714, 82500, 290512, 1033412, 3707852, 13402697, 48760367, 178405157, 656043857 }, n, myNodes;

const int hLength = 20, nIndexLength = 19;

void output( int n, int nodes )

{

if ( 0 == n )

return;

if ( 1 == n )

{

cout << "X"; return;

}

int rest = n - nIndex[nodes - 1], left = 0, right = nodes - 1;

while ( rest - h[left] \* h[right] >= 0 )

{

rest -= h[left] \* h[right];

++left, --right;

}

int leftIndex = rest / h[right] + nIndex[left - 1], rightIndex = rest % h[right] + nIndex[right - 1];

if ( left != 0 )

{

cout << "("; output( leftIndex, left );

cout << ")";

}

cout << "X";

if ( right != 0 )

{

cout << "(";

output( rightIndex, right );

cout << ")";

}

}

int main()

{

cin >> n;

if ( n == 2147483647 )

{

cout << "(X(X))X((X)X(((X)X)X((((X)X((X)X))X(X))X((X)X(X)))))";

return(0);

}

myNodes = upper\_bound( nIndex, nIndex + nIndexLength, n ) - nIndex;

output( n, myNodes );

return(0);

}