#include<stdio.h> #include<string.h> #include<stdlib.h>

#include <time.h> time\_t t1 = time(NULL); //print 秒數

**<math.h>**以下為一般數值計算工作的函數

|  |  |  |
| --- | --- | --- |
| **函數名稱** | **功能** | **函數原型** |
| fabs | 求絕對值 | [double fabs(double);](http://pydoing.blogspot.com/2010/07/c-fabs.html) |
| fmax | 求 x 與 y 之中的最大值 | [double fmax(double, double);](http://pydoing.blogspot.com/2010/07/c-fmax.html) |
| fmin | 求 x 與 y 之中的最小值 | [double fmin(double, double);](http://pydoing.blogspot.com/2010/07/c-fmin.html) |
| remainder | 求浮點餘數 | [double remainder(double, double);](http://pydoing.blogspot.com/2010/07/c-remainder.html) |
| fma | 求 (x \* y) + z | [double fma(double, double, double);](http://pydoing.blogspot.com/2010/07/c-fma.html) |
| round | 四捨五入到整數位 | [double round(double);](http://pydoing.blogspot.com/2010/07/c-round.html) |

以下為指數相關的函數

|  |  |  |
| --- | --- | --- |
| **函數名稱** | **功能** | **函數原型** |
| sqrt | 求平方根 | [double sqrt(double);](http://pydoing.blogspot.com/2010/07/c-sqrt.html) |
| cbrt | 求立方根 | [double cbrt(double);](http://pydoing.blogspot.com/2010/07/c-cbrt.html) |
| pow | 求 x 的 y 次方 | [double pow(double, double);](http://pydoing.blogspot.com/2010/07/c-pow.html) |

**字元測試 <ctype.h>**

以下為字元測試的函數

|  |  |  |
| --- | --- | --- |
| **函數名稱** | **功能** | **函數原型** |
| isdigit | 測試字元是否為數字 | [int isdigit(int);](http://pydoing.blogspot.com/2010/07/c-isdigit.html) |
| isalpha | 測試字元是否為字母 | [int isalpha(int);](http://pydoing.blogspot.com/2010/07/c-isalpha.html) |
| isalnum | 測試字元是否為數字或字母 | [int isalnum(int);](http://pydoing.blogspot.com/2010/07/c-isalnum.html) |
| isxdigit | 測試字元是否為十六進位數字 | [int isxdigit(int);](http://pydoing.blogspot.com/2010/07/c-isxdigit.html) |
| islower | 測試字元是否為小寫字母 | [int islower(int);](http://pydoing.blogspot.com/2010/07/c-islower.html) |
| isupper | 測試字元是否為大寫字母 | [int isupper(int);](http://pydoing.blogspot.com/2010/07/c-isupper.html) |
| isascii | 測試字元是否為 ASCII | [int isascii(int);](http://pydoing.blogspot.com/2010/07/c-isascii.html) |
| isblank | 測試是否為空白字元 | [int isblank(int);](http://pydoing.blogspot.com/2010/07/c-isblank.html) |
| isspace | 測試字元是否為空格 | [int isspace(int);](http://pydoing.blogspot.com/2010/07/c-isspace.html) |
| iscntrl | 測試是否為控制字元 | [int iscntrl(int);](http://pydoing.blogspot.com/2010/07/c-iscntrl.html) |
| ispunct | 測試是否為空格、數字、字母以外的可列印字元 | [int ispunct(int);](http://pydoing.blogspot.com/2010/07/c-ispunct.html) |
| isprint | 測試是否為含括空格以內的可列印字元 | [int isprint(int);](http://pydoing.blogspot.com/2010/07/c-isprint.html) |
| isgraph | 測試是否為空格以外的可列印字元 | [int isgraph(int);](http://pydoing.blogspot.com/2010/07/c-isgraph.html) |

以下為英文字母大、小寫轉換的函數

|  |  |  |
| --- | --- | --- |
| **函數名稱** | **功能** | **函數原型** |
| tolower | 將大寫字母轉換為小寫 | [int tolower(int);](http://pydoing.blogspot.com/2010/07/c-tolower.html) |
| toupper | 將小寫字母轉換為大寫 | [int toupper(int);](http://pydoing.blogspot.com/2010/07/c-toupper.html) |

以下函數可以拷貝字串

|  |  |  |
| --- | --- | --- |
| **函數名稱** | **功能** | **函數原型** |
| strcpy | 將字串 s2 拷貝到 s1 | [char \*strcpy(char \*s1, const char \*s2);](http://pydoing.blogspot.com/2010/07/c-strcpy.html) |
| strncpy | 將字串 s2 最多 n 個字元拷貝到 s1 | [char \*strncpy(char \*s1, const char \*s2, size\_t n);](http://pydoing.blogspot.com/2010/07/c-strncpy.html) |

以下函數可以將字串相接

|  |  |  |
| --- | --- | --- |
| **函數名稱** | **功能** | **函數原型** |
| strcat | 將字串 s2 接到 s1 的尾端 | [char \*strcat(char \*s1, const char \*s2);](http://pydoing.blogspot.com/2010/07/c-strcat.html) |
| strncat | 將字串 s2 最多 n 個字元接到 s1 的尾端 | [char \*strncat(char \*s1, const char \*s2, size\_t);](http://pydoing.blogspot.com/2010/07/c-strncat.html) |

以下函數測試兩個字串是否相等

|  |  |  |
| --- | --- | --- |
| **函數名稱** | **功能** | **函數原型** |
| strcmp | 比較 s1 與 s2 兩個字串是否相等 | [int strcmp(const char \*s1, const char \*s2);](http://pydoing.blogspot.com/2010/07/c-strcmp.html) |
| strncmp | 比較 s1 與 s2 兩個字串前 n 個字元是否相等 | [int strncmp(const char \*s1, const char \*s2, size\_t n);](http://pydoing.blogspot.com/2010/07/c-strncmp.html) |

以下函數作為字串的搜尋處理之用

|  |  |  |
| --- | --- | --- |
| **函數名稱** | **功能** | **函數原型** |
| strchr | 回傳在字串 s 中，字元 c 第一次出現位置的指標 | [char \*strchr(const char \*s, int c);](http://pydoing.blogspot.com/2010/07/c-strchr.html) |
| strcspn | 計算經過幾個字元會在字串 s1 中遇到屬於 s2 中的字元 | [size\_t strcspn(const char \*s1, const char \*s2);](http://pydoing.blogspot.com/2010/07/c-strcspn.html) |
| strspn | 計算經過幾個字元會在字串 s1 中遇到不屬於 s2 中的字元 | [size\_t strspn(const char \*s1, const char \*s2);](http://pydoing.blogspot.com/2010/07/c-strspn.html) |
| strpbrk | 回傳在字串 s2 中的任何字元在 s1 第一次出現位置的指標 | [char \*strpbrk(const char \*s1, const char \*s2);](http://pydoing.blogspot.com/2010/07/c-strpbrk.html) |
| strrchr | 回傳在字串 s 中，字元 c 最後一次出現位置的指標 | [char \*strrchr(const char \*s, int c);](http://pydoing.blogspot.com/2010/07/c-strrchr.html) |
| strstr | 回傳在字串 s2 在 s1 第一次出現位置的指標 | [char \*strstr(const char \*s1, const char \*s2);](http://pydoing.blogspot.com/2010/07/c-strstr.html) |
| strtok | 以字串 s2 的內容切割 s1 | [char \*strtok(char \*s1, const char \*s2);](http://pydoing.blogspot.com/2010/07/c-strtok.html) |

以下函數計算字串的長度

|  |  |  |
| --- | --- | --- |
| **函數名稱** | **功能** | **函數原型** |
| strlen | 計算字串的長度 | [size\_t strlen(const char \*s);](http://pydoing.blogspot.com/2010/07/c-strlen.html) |

以下函數為進行記憶體區塊操作之用

|  |  |  |
| --- | --- | --- |
| **函數名稱** | **功能** | **函數原型** |
| memcpy | 從 s2 所指向的資料複製 n 個字元到 s1 | [void \*memcpy(void \*s1, const void \*s2, size\_t n);](http://pydoing.blogspot.com/2010/07/c-memcpy.html) |
| memmove | 從 s2 所指向的資料複製 n 個字元到 s1 | [void \*memmove(void \*s1, const void \*s2, size\_t n);](http://pydoing.blogspot.com/2010/07/c-memmove.html) |
| memcmp | 比較 s1 與 s2 前 n 個字元的資料 | [int memcmp(const void \*s1, const void \*s2, size\_t n);](http://pydoing.blogspot.com/2010/07/c-memcmp.html) |
| memchr | 找出字元 c 在 s 前 n 個字元第一次出現的位置 | [void \*memchr(const void \*s, int c, size\_t n);](http://pydoing.blogspot.com/2010/07/c-memchr.html) |
| memset | 將 s 中前 n 個字元全部設定為 c | [void \*memset(void \*s, int c, size\_t n);](http://pydoing.blogspot.com/2010/07/c-memset.html) |

以下函數為字元及字串的輸出入

|  |  |  |
| --- | --- | --- |
| **函數名稱** | **功能** | **函數原型** |
| getchar | 從標準輸入裝置讀入一個字元 | [int getchar(void);](http://pydoing.blogspot.com/2010/07/c-getchar.html) |
| gets | 從標準輸入裝置讀入字串 | [char \*gets(char \*s);](http://pydoing.blogspot.com/2010/07/c-gets.html) |
| putchar | 將字元傳送到標準輸出裝置 | [int putchar(int c);](http://pydoing.blogspot.com/2010/07/c-putchar.html) |
| puts | 將字串傳送到標準輸出裝置 | [int puts(const char \*s);](http://pydoing.blogspot.com/2010/07/c-puts.html) |

以下函數為格式化輸出

|  |  |  |
| --- | --- | --- |
| **函數名稱** | **功能** | **函數原型** |
| printf | 將格式化字串傳送到標準輸出裝置 | [int printf(const char \*format, ...);](http://pydoing.blogspot.com/2010/07/c-printf.html) |
| sprintf | 將格式化字串傳送到陣列 | [int sprintf(char \*s, const char \*format, ...);](http://pydoing.blogspot.com/2010/07/c-sprintf.html) |

以下函數為格式化輸入

|  |  |  |
| --- | --- | --- |
| **函數名稱** | **功能** | **函數原型** |
| scanf | 從標準輸入裝置擷取格式化輸入 | [int scanf(const char \*format, ...);](http://pydoing.blogspot.com/2010/07/c-scanf.html) |
| sscanf | 從陣列讀取格式化字串 | [int sscanf(char \*s, const char \*format, ...);](http://pydoing.blogspot.com/2010/07/c-sscanf.html) |

以下函數可開啟檔案、刪除檔案或更改檔案名稱

|  |  |  |
| --- | --- | --- |
| **函數名稱** | **功能** | **函數原型** |
| fopen | 回傳指定檔案的資訊給 FILE 型態的結構指標 | [FILE \*fopen(const char \*filename, const char \*mode);](http://pydoing.blogspot.com/2010/07/c-fopen.html) |
| freopen | 回傳指定檔案的資訊給 FILE 型態的結構指標，並重導串流的方向 | [FILE \*freopen(const char \*filename, const char \*mode, FILE \*stream);](http://pydoing.blogspot.com/2010/07/c-freopen.html) |
| fflush | 強制將緩衝區中的輸出串流寫到檔案中 | [int fflush(FILE \*stream);](http://pydoing.blogspot.com/2010/07/c-fflush.html) |
| fclose | 將緩衝區中為輸出資料輸出，清除輸入資料，接著關閉串流 | [int fclose(FILE \*stream);](http://pydoing.blogspot.com/2010/07/c-fclose.html) |
| remove | 刪除檔案 | [int remove(const char \*filename);](http://pydoing.blogspot.com/2010/07/c-remove.html) |
| rename | 將檔案更名 | [int rename(const char \*oldname, const char \*newname);](http://pydoing.blogspot.com/2010/07/c-rename.html) |

以下函數進行檔案的格式化字串處理

|  |  |  |
| --- | --- | --- |
| **函數名稱** | **功能** | **函數原型** |
| fprintf | 將格式化字串輸出到檔案 | [int fprintf(FILE \*stream, const char \*format, ...);](http://pydoing.blogspot.com/2010/07/c-fprintf.html) |
| fscanf | 從檔案輸入格式化字串 | [int fscanf(FILE \*stream, const char \*format, ...);](http://pydoing.blogspot.com/2010/07/c-fscanf.html) |

以下函數進行檔案的字元及字串的處理

|  |  |  |
| --- | --- | --- |
| **函數名稱** | **功能** | **函數原型** |
| fgetc | 從檔案中取得字元 | [int fgetc(FILE \*stream);](http://pydoing.blogspot.com/2010/07/c-fgetc.html) |
| fgets | 從檔案中取得單行字串 | [char \*fgets(char \*s, int n, FILE \*stream);](http://pydoing.blogspot.com/2010/07/c-fgets.html) |
| fputc | 把字元放到檔案中 | [int fputc(const char \*s, FILE \*stream);](http://pydoing.blogspot.com/2010/07/c-fputc.html) |
| fputs | 把字串放到檔案中 | [int fputs(const char \*s, FILE \*stream);](http://pydoing.blogspot.com/2010/07/c-fputs.html) |

以下函數利用陣列進行檔案的存取

|  |  |  |
| --- | --- | --- |
| **函數名稱** | **功能** | **函數原型** |
| fread | 將檔案中具有 nobj 個元素，每個元素大小為 size 的資料讀進陣列 ptr 中 | [size\_t fread(void \*ptr, size\_t size, siz\_t nobj, FILE \*stream)](http://pydoing.blogspot.com/2010/07/c-fread.html) |
| fwrite | 將陣列 ptr 中具有 nobj 個元素，每個元素大小為 size 的資料寫到檔案中 | [size\_t fwrite(const void \*ptr, size\_t size, size\_t nobj, FILE \*stream)](http://pydoing.blogspot.com/2010/07/c-fwrite.html) |

以下函數處理檔案可存取的位置

|  |  |  |
| --- | --- | --- |
| **函數名稱** | **功能** | **函數原型** |
| fseek | 設定檔案資料存取的位置 | [int fseek(FILE \*stream, long offset, int origin);](http://pydoing.blogspot.com/2010/07/c-fseek.html) |
| ftell | 回傳目前檔案資料可存取的位置 | [long ftell(FILE \*stream);](http://pydoing.blogspot.com/2010/07/c-ftell.html) |
| rewind | 將檔案存取位置重新設為檔案開頭 | [void rewind(FILE \*stream);](http://pydoing.blogspot.com/2010/07/c-rewind.html) |

河內塔

#include <stdio.h>

void hanoi(int n, char A, char B, char C){

if(n == 1) { printf("Move sheet from %c to %c\n", A, C); }

else { hanoi(n-1, A, C, B);

hanoi(1, A, B, C);

hanoi(n-1, B, A, C); } }

int main() {

int n;

printf("請輸入盤數：");

scanf("%d", &n);

hanoi(n, 'A', 'B', 'C');

return 0; }

### [Binary Search Tree](http://squall.cs.ntou.edu.tw/cpp/103spring/labtest/test1/BinarySearchTree.html)

struct binary\_search\_tree\_node

{

int data;

struct binary\_search\_tree\_node \*left;

struct binary\_search\_tree\_node \*right;

};

typedef struct binary\_search\_tree\_node BSTNode;

void print\_inorder(BSTNode \*ptr);

BSTNode \*insert\_node(BSTNode \*root, int value);

BSTNode \*find\_node(BSTNode \*ptr, int value);

int main()

{

int data[] = {10, 20, 5, 8, 30, 15, 1, 18};

int ndata = sizeof(data)/sizeof(int);

BSTNode \*head = 0;

for (int i=0; i<ndata; i++)

head = insert\_node(head, data[i]);

printf("In order listing: ");

print\_inorder(head);

printf("\n"); fflush(stdout);

assert(find\_node(head, 15));

assert(!find\_node(head, 11));

system("pause");

return 0;

}

// 中序走訪

void print\_inorder(BSTNode \*ptr)

{

if (ptr != 0)

{

print\_inorder(ptr->left); // 走左子樹

printf("%2d ", ptr->data); // 印出節點內容

print\_inorder(ptr->right); // 走右子樹

}

}

// 插入節點

BSTNode \*insert\_node(BSTNode \*root, int value)

{

BSTNode \*new\_node;

BSTNode \*current;

BSTNode \*parent;

// 建立節點

new\_node = (BSTNode \*) malloc(sizeof(BSTNode));

new\_node->data = value;

new\_node->left = 0;

new\_node->right = 0;

if (root == 0) // 目前無資料

{

return new\_node;

}

else

{

current = root; // 從頭找要新節點之插入點

while (current != 0)

{

parent = current; // 找新節點之父節點

if (current->data > value)

current = current->left; // 往左找

else

current = current->right; // 往右找

}

if (parent->data > value) // 插入此父節點左邊或右邊

parent->left = new\_node;

else

parent->right = new\_node;

}

return root; // 回傳此樹

}

@malloc() 用來做動態記憶體配置，產生動態的記憶體空間，需要一個利用 sizeof 計算所佔記憶體空間大小的參數。若建立成功， malloc() 回傳指向該空間的指標，否則回傳 NULL 。

int \*nPtr;

nPtr = malloc(sizeof(int));

if (nPtr == NULL) {

printf("建立記憶體空間失敗\n");

}

@ free() 用來做動態記憶體配置，釋放原先所建立指向的記憶體空間

free(tPtr);

@將字串中的數字轉換為 int

 printf("%d\n", atoi("54321"));

@stdlib.h 的函數 qsort() 替陣列進行排序，共需四個參數。第一個參數為所欲排序的陣列，第二個參數為該陣列的個數，第三個參數為利用 sizeof 計算陣列元素所佔的記憶體空間，第四個參數為函數指標，須自行定義排序陣列的排序比較方式。

int cmp(const void \*s1, const void \*s2);

int main(void)

{

char test[] = "qwertyuioplkjhgfdsazxcvbnm";

qsort(test, 26, sizeof(char), cmp);

printf("%s\n", test);

return 0;

}

int cmp(const void \*s1, const void \*s2)

{

return \*(char \*)s1 - \*(char \*)s2;

}

@stdlib.h 的函數 srand() 替[rand()](http://pydoing.blogspot.com/2010/07/c-rand.html)產生種子，通常以 time.h 的 [time()](http://pydoing.blogspot.com/2010/07/c-time.html)當作參數，可以使每次產生的擬隨機數都不同。

int main(void)

{

int i;

srand(time(0));

for (i = 1; i <= 100; i++) {

printf("%d ", rand());

if (i % 5 == 0) {

printf("\n");

}

}

return 0;

}

Backtracking

中文稱作「回溯法」，枚舉多維度數值的方法。運用遞迴依序窮舉各個維度的數值，製作所有可能的多維度數值，並且在遞迴途中避免枚舉不正確的多維度數值。

1. int solution[MAX\_DIMENSION];    // 多維度數值
2. void backtrack(int dimension){
3. /\* prune：在遞迴途中避免枚舉不正確的多維度數值 \*/
4. if ( solution[] will not be an answer ) return;
5. /\* 製作了一組多維度數值，並檢驗正不正確 \*/
6. if ( dimension == MAX\_DIMENSION ){
7. check and record solution[];
8. return;
9. }
10. /\* 窮舉這個維度的所有數值，並遞迴到下一個維度 \*/
11. for ( x = possible value of current dimension ){
12. solution[dimension] = x;
13. backtrack( dimension + 1 );
14. }
15. }
16. int main(){
17. backtrack(0);   // 從第一個維度開始枚舉
18. }

**Enumerate Permutations**

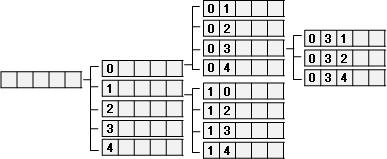
Permutation

便是數學課本中「排列組合」的「排列」。但是這裡不是要計算排列有多少種，而是枚舉所有的排列，以字典順序枚舉。

例如 {1,2,3} 所有的排列就是 {1,2,3} 、 {1,3,2} 、 {2,1,3} 、 {2,3,1} 、 {3,1,2} 、 {3,2,1} 。

範例：枚舉 {0,1,2,3,4} 所有排列

依序枚舉每個位置。針對每個位置，試著填入各種數字。



1. void backtrack(int n)
2. {
3. // it's a solution
4. if (n == 5)
5. {
6. for (int i=0; i<5; i++)
7. cout << solution[i] << ' ';
8. cout << endl;
10. return;
11. }
13. // 寫成一個迴圈
14. for (int i=0; i<5; i++)
15. if (!used[i])
16. {
17. used[i] = true;
19. solution[n] = i;
20. backtrack(n+1);
22. used[i] = false;
23. }
24. }

**Enumerate Combinations**

Combination （ Subset ）

便是數學課本中「排列組合」的「組合」；概念等於「子集合」。但是這裡不是要計算組合有多少種，而是枚舉所有的組合，以字典順序枚舉。

例如 {1,2,3} 所有的組合就是 {} 、 {1} 、 {2} 、 {3} 、 {1,2} 、 {1,3} 、 {2,3} 、 {1,2,3} 。

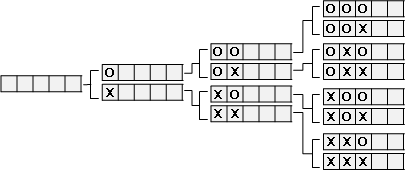
範例：枚舉 {0,1,2,3,4} 所有組合

該如何枚舉呢？先觀察平時我們計算組合個數的方法。

{0,1,2,3,4} 所有組合個數總共 2^5 個： 0 可取可不取，有兩種情形、 1 可取可不取，有兩種情形、 ... 、 4 可取可不取，有兩種情形。根據乘法原理，總共 2\*2\*2\*2\*2 = 2^5 種情形。

枚舉方式可以仿照乘法原理。建立一個陣列，當作一個集合。 solution[i] = true 表示這個集合擁有第 i 個元素，觀念等同「 [Set 資料結構 : 索引儲存](http://www.csie.ntnu.edu.tw/~u91029/Set.html)」。

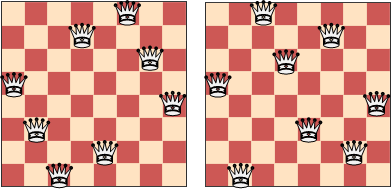
依序枚舉每個位置。針對每個位置，試著填入取或不取。



1. bool solution[5];   // 索引儲存
3. void backtrack(int n)
4. {
5. // it's a solution
6. if (n == 5)
7. {
8. for (int i=0; i<5; i++)
9. if (solution[i])
10. cout << i << ' ';
11. cout << endl;
12. return;
13. }
15. // 選取數字n，然後繼續枚舉之後的位置。
16. solution[n] = true;
17. backtrack(n+1);
19. // 不取數字n，然後繼續枚舉之後的位置。
20. solution[n] = false;
21. backtrack(n+1);
22. }
24. void enumerate\_combinations()
25. {
26. backtrack(0);
27. }

**8 Queen Problem**

八皇后問題



問題：在 8x8 的西洋棋棋盤上擺放八隻皇后，讓他們恰好無法互相攻擊對方。皇后的攻擊範圍是米字。

一個非常簡單的想法：每一格都有「放」和「不放」兩種選擇，枚舉所有可能，並避免枚舉皇后互相攻擊的情形。建立 8x8 的 bool 陣列，代表一個 8x8 的棋盤盤面情形。例如 solution[0][0] = true 表示 (0,0) 這個位置有放置皇后。

1. int solution[8];
3. void backtrack(int x)   // 每次都換一排格子
4. {
5. // it's a solution
6. if (x == 8)
7. {
8. print\_solution();
9. return;
10. }
12. // 分別放置皇后在每一格，並各自遞迴下去。
13. for (int y=0; y<8; ++y)
14. {
15. solution[x] = y;
16. backtrack(x+1);
17. }
18. }

接著要避免枚舉不可能出現的答案。

1. int solution[8];
2. bool my[8], md1[15], md2[15];   // 初始值都是 false
3. // x這條線可以不用檢查了
5. void backtrack(int x)   // 每次都換一排格子
6. {
7. // it's a solution
8. if (x == 8)
9. {
10. print\_solution();
11. return;
12. }
14. // 分別放置皇后在每一格，並各自遞迴下去。
15. for (int y=0; y<8; ++y)
16. {
17. int d1 = (x+y) % 15, d2 = (x-y+15) % 15;
19. if (!my[y] && !md1[d1] && !md2[d2])
20. {
21. // 這隻皇后佔據了四條線，記得標記起來。
22. my[y] = md1[d1] = md2[d2] = true;
24. solution[x] = y;
25. backtrack(x+1);
27. // 遞迴結束，回復到原本的樣子，要記得取消標記。
28. my[y] = md1[d1] = md2[d2] = false;
29. }
30. }
31. }

改進

8 Queen Problem 的答案是上下、左右、對角線對稱的。排除對稱的情形，可以節省枚舉的時間。這裡不加贅述。

另一種左右斜線判斷方式

比用陣列記錄還麻煩。自行斟酌。

1. void backtrack(int x)   // 每次都換一排格子
2. {
3. for (int i=0; i<x; ++i)
4. if (abs(x - i) == abs(solution[x] - solution[i]))
5. return;
7. ......
8. }

UVa [167](http://uva.onlinejudge.org/external/1/167.html) [750](http://uva.onlinejudge.org/external/7/750.html) [10513](http://uva.onlinejudge.org/external/105/10513.html) [639](http://uva.onlinejudge.org/external/6/639.html) [989](http://uva.onlinejudge.org/external/9/989.html) [10893](http://uva.onlinejudge.org/external/108/10893.html) [10957](http://uva.onlinejudge.org/external/109/10957.html)

**0/1 Knapsack Problem**

0/1 背包問題



問題：各式各樣的物品，重量與價值皆異。一個背包，具有耐重限制。現在將物品儘量塞入背包，令背包裡物品總價值最高。

一個簡單的想法：每個物品都有「要」和「不要」兩種選擇，窮舉所有可能，並避免枚舉背包超載的情形。建立一維 bool 陣列， solution[0] = true 表示第零個物品有放進背包。

1. bool solution[10];  // 十個物品
3. int weight[10] = {4, 54, 1, ..., 32};   // 十個物品分別的重量
4. int cost[10] = {3, 3, 11, ..., 23};     // 十個物品分別的價值
6. const int maxW = 100;   // 背包承載上限
7. int maxC = 0;           // 出現過的最高總值
9. void backtrack(int n, int w, int c)
10. {
11. // it's a solution
12. if (n == 10)
13. {
14. if (c > maxC)   // 記錄總值最高的
15. {
16. maxC = c;
17. store\_solution();
18. }
19. return;
20. }
22. // 放進背包
23. if (w + weight[n] <= maxW)  // 檢查背包超載
24. {
25. solution[n] = true;
26. backtrack(n+1, w + weight[n], c + cost[n]);
27. }
29. // 不放進背包
30. solution[n] = false;
31. backtrack(n+1, w, c);
32. }

35. bool answer[10];    // 正確答案
37. void store\_solution()
38. {
39. for (int i=0; i<10; ++i)
40. answer[i] = solution[i];
41. }

檢查背包超載的部分可以修改成更美觀的樣子。

1. void backtrack(int n, int w, int c)
2. {
3. if (w > maxW) return;   // 背包超載
5. // it's a solution
6. if (n == 10)
7. {
8. if (c > maxC)   // 記錄總值最高的
9. {
10. maxC = c;
11. store\_solution();
12. }
13. return;
14. }
16. // 放進背包
17. solution[n] = true;
18. backtrack(n+1, w + weight[n], c + cost[n]);
20. // 不放進背包
21. solution[n] = false;
22. backtrack(n+1, w, c);
23. }

各位可以預先排序物品重量，再執行 backtracking 程式碼，看看效率有何不同。然後嘗試使用 heuristic bound 加快速度。

**Inclusion-Exclusion Principle**

排容原理



類似於枚舉所有子集合，但是每個子集合有正負號之別 ── 奇數個集合的交集為正號、偶數個集合的交集為負號。

舉例：求出 1 到 100 當中不可被 3 或 5 或 8 整除的整數有幾個。 3 、 5 、 8 均兩兩互質。



1. int array[3] = {3, 5, 8};
3. // 排容，sign為正負號，divisor為各種可能的除數
4. int backtrack(int n, int sign, int divisor)
5. {
6. // it's a solution
7. if (n == 3) return sign \* (100 / divisor);
9. int total = 0;
11. // 不選。正負號維持不變，除數維持不變。
12. // solution[n] = false;
13. total += backtrack(n+1, sign, divisor);
15. // 選。須變號，並逐步累計除數。
16. // 因逐步累計除數，故不需要具體記錄選到的數字
17. // solution[n] = true;
18. total += backtrack(n+1, -sign, divisor \* array[n]);
20. return total;
21. }
23. void inclusion\_exclusion()
24. {
25. cout << "1到100當中不可被3或5或8整除的整數";
26. cout << "有" << backtrack(0, +1, 1) << "個";
27. }

考慮數字之間不互質的一般情形：

1. int array[5] = {3, 5, 6, 7, 9};
3. // 最大公因數
4. int gcd(int a, int b)
5. {
6. return b ? gcd(b, a%b) : a;
7. }
9. // 最小公倍數
10. int lcm(int a, int b)
11. {
12. return a / gcd(a, b) \* b;
13. }
15. // 精簡過後的排容程式碼，divisor為各種可能的除數
16. int backtrack(int n, int divisor)
17. {
18. if (n == 5) return 100 / divisor;
19. return backtrack(n+1, divisor)
20. - backtrack(n+1, lcm(divisor, array[n]));
21. }
23. void inclusion\_exclusion()
24. {
25. cout << "1到100當中不可被3或5或6或7或9整除的整數";
26. cout << "有" << backtrack(0, 1) << "個";
27. }

枚舉子集合（組合）有兩種枚舉方式，排容原理亦有兩種枚舉方式。

1. int array[5] = {3, 5, 6, 7, 9};
3. int backtrack(int n, int divisor)
4. {
5. int total = 0;
6. total += 100 / divisor; // 目前湊出來的集合
8. // 繼續枚舉之後的數字，記得變號
9. for (int i=n; i<5; ++i)
10. total -= backtrack(i+1, lcm(divisor, array[i]));
12. return total;
13. }
15. void inclusion\_exclusion()
16. {
17. cout << "1到100當中不可被3或5或6或7或9整除的整數";
18. cout << "有" << backtrack(0, 1) << "個";
19. }

UVa [10325](http://uva.onlinejudge.org/external/103/10325.html)

**Euclidean Shortest Path**

© 2010 [tkcn](http://tkcnandy.blogspot.com/) . All rights reserved.

500 ms Next Play Stop Reset