Câu 1. Tên bài: **TOURISM.CPP**

Hành trình cần tìm có dạng $(x_1 = 1, x_2, \ldots, x_n, x_{n+1} = 1)$ ở đây giữa x_i và x_{i+1}: hai thành phố liên tiếp trong hành trình phải có đường đi trực tiếp ($C_{ij} \neq +\infty$) và ngoại trừ thành phố 1, không thành phố nào được lặp lại hai lần. Có nghĩa là dãy $(x_1, x_2, \ldots, x_n)$ lập thành 1 hoán vị của $(1, 2, \ldots, n)$.

Duyệt quay lui: x_2 có thể chọn một trong các thành phố mà x_1 có đường đi tới (trực tiếp), với mỗi cách thử chọn x_2 như vậy thì x_3 có thể chọn một trong các thành phố mà x_2 có đường đi tới (ngoài x_1). Tổng quát: x_i có thể chọn 1 trong các thành phố chưa đi qua mà từ x_{i-1} có đường đi trực tiếp tới ($1 \leq i \leq n$).

Nhánh cận: Khởi tạo cấu hình *BestConfig* có chi phí = $+\infty$. Với mỗi bước thử chọn x_i xem chi phí đường đi cho tới lúc đó có < Chi phí của cấu hình *BestConfig* ?, nếu không nhỏ hơn thì thử giá trị khác ngay bởi có đi tiếp cũng chỉ tốn thêm. Khi thử được một giá trị x_n ta kiểm tra xem x_n có đường đi trực tiếp về 1 không ?. Nếu có hãy đánh giá chi phí đi từ thành phố 1 đến x_n cộng với chi phí từ x_n đi đi trực tiếp về 1, nếu nhỏ hơn chi phí của đường đi *BestConfig* thì cập nhật lại *BestConfig* bằng cách đi mới.

Sau thủ tục tìm kiếm quay lui mà chi phí của *BestConfig* vẫn bằng $+\infty$ thì có nghĩa là nó không tìm thấy một hành trình nào thoả mãn điều kiện đề bài để cập nhật *BestConfig*, bài toán không có lời giải, còn nếu chi phí của *BestConfig* < $+\infty$ thì in ra cấu hình *BestConfig* - đó là hành trình ít tốn kém nhất tìm được

Câu 2. Tên bài: **CAITUI.CPP**

Ta sẽ sinh hết tất cả các cấu hình, với gói hang thứ *i* thì tên trộm có thể lấy hoặc không lấy. Tuy vậy, ta có thể thêm nhánh cận vào để giảm thời gian chương trình như sau :

+ Nếu trong cấu hình đang xây dựng, tổng $W > m$ thì không cần xây dựng tiếp vì nó không thỏa mãn

+ Sau khi xây dựng được một vài cấu hình, ta được 1 kết quả tạm thời là *res*. Nếu trong cấu hình tiếp theo, khi xây dựng đến gói hàng thứ *i* và có tổng giá trị lấy được từ $1 \rightarrow i - 1$ là *val*, thì *val + sum_val(i , i + 1, ... , n) <= res* thì không cần xây dựng nữa. Trong code dưới *sum[i]* là tổng các giá trị của các gói hàng *i, i+1, ...n*

Câu 3. Tên bài: **NGOAC.CPP**

Dùng thuật toán quay lui với mảng *x* đếm số lượng ngoặc mở và ngoặc đóng. Ta xem ngoặc mở là 0, ngoặc đóng là 1. Tại mỗi bước chọn phải đảm bảo rằng số lượng của dấu ngoặc đó phải nhỏ hơn hoặc bằng *n div 2*. Đó là cận thứ nhất. Tiếp theo phải đảm bảo rằng sau mỗi bước chọn số lượng ngoặc mở phải lớn hoặc bằng số lượng ngoặc đóng. Đó là cận thứ hai.

Câu 4. Tên bài: **TRES.CPP**

Mình sẽ sinh tất cả cấu hình các món ăn , sau đó thử từng người khách xem có thỏa mãn không. Đặt cận ở chỗ ta xét nếu người khách nào mà có số món ăn yêu thích > 2 thì không cần xét tiếp cấu hình đó nữa. Thứ hai, nếu tổng số tiền đang xây dựng ở cấu hình này lớn hơn số tiền của cấu hình thỏa mãn trước đó thì không cần xây dựng nữa.

Ta thấy vì mỗi người có số lượng món ăn yêu thích là bất kì (không biết trước số lượng phần tử của tệp) nên khi đọc dữ liệu, ta sẽ dùng hàm *getline(...)* một xâu với mỗi người, sau đó chuyển từ xâu đó sang các số để dễ xử lí.

Câu 5. Tên bài: **MINE.CPP**

- Xét một ô, nếu 7 trong 8 ô xung quanh nó đều đã biết có mìn hay không thì có thể suy ra được ô còn lại có mìn hay không. Giả sử đã biết được toàn bộ các ô hàng trên cùng và hàng ngoài cùng bên trái có mìn hay không (dùng đệ quy quay lui để tạo ra giả thiết này)
- Xét ô đặc biệt là ô ở góc trái trên (1, 1), xung quanh nó chỉ có 3 ô, như vậy nếu biết được 2 ô xung quanh: 1 ô nằm ở hàng trên cùng (1, 2), 1 ô nằm ở hàng ngoài cùng bên trái (2, 1) thì ta có thể suy ra được ô thứ ba (1, 1). Lúc này ô (2, 2) lại đóng vai trò như ô (1, 1) ban nãy do đó ta tìm được (3, 2).
- Làm lần lượt ta tìm được toàn bộ bảng. Nếu có một bước nào đó không thực hiện được thì chứng tỏ giả thiết là sai, tiếp tục thực hiện đệ quy quay lui.
- ***Nhánh cận*** : Để giảm tính toán ta không nhất thiết phải tính toàn bộ hàng trên cùng và hàng ngoài cùng rồi mới suy ra toàn bảng để kiểm tra mà có thể vừa tính vừa suy, cụ thể suy ra toàn bộ hàng ngoài cùng bên trái, sau đó mỗi lần chạy đệ quy với một ô của hàng trên cùng ta xác định tất cả các ô cùng cột với nó có mìn hay không. Nếu ở bước nào đó không xác định được thì quay lui.