Bài 1

1.1.

- Sử dụng thuật toán DFS ta có thể tìm đường đi từ A đến B trong đồ thị có hướng G = (V, E).
- Độ phức tạp O(V + E) trong đó V là tập đỉnh, E là tập cạnh.
- Cách thức truy vết: trace[v] = u nếu ta đi từ u đến v
  trace[u] = -1 nếu ta bắt đầu từ u
- Xem code ở file bai1.1.py

1.3.

- Ở mỗi vị trí sẽ có 2 trạng thái 0/1, mỗi trạng thái sinh ra 1 số nhị phân. Trên mỗi trạng thái được sinh ra ta lại tiếp tục thay đổi từng vị trí để tạo trạng thái mới cho đến khi không tạo được nữa thì quay lại.
- Độ phức tạp $O(2^n)$
- Xem code ở file bai1.3.py

Bài 2

2.1

- Nhận thấy với dãy hoán vị của A1 .. An, các thành phần A_i sẽ lần lượt đứng ở vị trí đầu tiên và phía sau là dãy hoán vị của các số còn lại.
  → premutation(A) = $\Sigma(A_i$ + premutation(A\{A_i}))
- Độ phức tạp O(N!)
- Xem code ở file bai2.1.py

2.4

- Nếu dãy X_1 X_2 .. X_n thoả mãn 2 đoạn con bất kỳ liền nhau đều khác nhau, thì trong 4 ký tự liên tiếp bất kỳ bao giờ cũng phải có 1 ký tự “C”. Như vậy với một dãy gồm k ký tự liên tiếp của dãy X thì số ký tự C trong dãy con đó bắt buộc phải >= k/4.
  Tại bước thử chọn Xi, nếu ta đã có Ti ký tự “C” trong đoạn đã chọn từ X1 đến Xi, thì cho dù các bước đệ quy tiếp sau làm tốt như thế nào đi nữa, số ký tự “C” sẽ phải chọn thêm >= (N-i)/4 → loại bỏ nhánh này.
- Với N =100, chỉ mất khoảng 3 giây cho kết quả là xâu 27 ký tự “C”. Nếu không có đánh giá nhánh cận thì thời gian có thể nên hàng giờ.
- Xem code ở file bai2.4.py

Bài 3

3.1 : Backtracking : Quay lui

- Bài toán: Cho 1 mảng a nxm phần tử. Biết rằng trên mỗi ô có biểu thị các số nguyên. Nếu a[i][j] > 0 thì ô được phép đi qua, < 0 ô bị chặn, = 0 tại điểm 1, 1 hoặc n, m. Bạn chỉ được đi sang phải hoặc đi xuống dưới 1 ô so với ô hiện tại
  Yêu cầu: Tìm 1 đường đi bất kỳ từ ô 1, 1 đến n, m thoả mãn.
- Các làm: Ở 1 đỉnh i, j ta có 2 lựa chọn đi xuống i+1, j hoặc đi sang I, j+1. Ta đến thử từng ô kề để xem có thể đi trực tiếp từ ô hiện tại sang không, nếu không thể thì quay lại.
- Độ phức tạp O(2^(n+m)
- Xem code ở file bai3.1.py
- Upgrade: Nếu xét ngược lại bài toán, để tới ô a[i][j] chỉ có thể đi được từ 1 trong 2 ô a[i-1][j] hoặc a[i][j-1]. Ta hoàn toàn có thể xét tại ô a[i][j] có đường đi từ a[1][1] tới hay không nếu đã biết 2 ô kề có đường đi từ a[1][1] tới hay không.
  Độ phức tạp giảm xuống chỉ còn O(n*m)

3.2 : Branch and Bound : Nhánh cận

- Bài toán: Có thể định nghĩa khái niệm dãy ngoặc đúng dưới dạng đệ quy như sau
  1. () là dãy ngoặc đúng
  2. C là dãy ngoặc đúng nếu C = (A) hay C = AB với A, B là các dãy ngoặc đúng.

  Ví dụ dãy ngoặc đúng : (), (()), ()(), (())()
  Ví dụ dãy ngoặc sai:)(, ((, ()((,)()(
  Cho với n chẵn, in ra các dãy ngoặc đúng có chiều dài n. Tính số lượng.
- Cách làm: Ta sẽ sinh ‘(‘ trước. Với mỗi lần sinh ‘(‘ ta có thể tính được số lượng ‘)’ cần phải sinh và số lượng ô còn lại.
- Với thuật toán vét cạn thông thường, độ phức tạp là O(2^n). Với nhánh cận, ta giảm được đáng kể thời gian thao tác trên các nhánh không đem lại kết quả, vậy nên thời gian tính toán cũng được giảm đi (với n = 18, giảm từ 2s xuống 0.3s)
- Xem code ở file bai3.2.py