Contents

Flowchart:

1. Trang Web Vẽ?
⇨ Vào Link
⇨ https://app.diagrams.net/
⇨ Click biểu tượng “Google Drive” + Click “Ủy quyền” + chọn tài khoản Google để lưu biểu đồ + Click “Tiếp tục” + Click “Tạo biểu đồ mới” + Click “Tạo” + Click “No, pick folder…” + Vào Tab “Google Drive” + chọn thư mục muốn lưu biểu đồ, các thư mục được liệt kê ở đây không phân biệt cấp cha con + Click “Chọn” + khi này bạn sẽ vào cửa sổ Edit, đồng thời cũng sẽ có File với phần mở rộng “.drawio” được tạo trong Google Drive
⇨ Nội dung trong cửa sổ Edit sẽ tự động được lưu vào Google Drive liên tục, kể cả tên File
⇨ 1 cửa sổ Edit chỉ chỉnh sửa được 1 File
2. Cấu Trúc 1 File?
⇨ Gồm nhiều trang

3. Đổi Tên File?

⇨ Click vào tên File góc trái trên + nhập tên mới + Click “Đổi tên”

4. Tạo Thêm 1 Trang?

⇨ Click dấu “+” ở thanh bên dưới bên phải các Tab trang

5. Đổi Tên Trang?

⇨ Phải chuột vào Tab trang + chọn “Đổi tên…” + nhập tên mới + Click “Đổi tên”

6. Xuất Trang Hiện Tại Dưới Dạng Hình Ảnh?

⇨ Vào Tab “Tập tin” + chọn “Xuất tập tin dưới dạng” + chọn “PNG…” + chọn “Xuất tập tin” + tại mục “Lưu dưới dạng” + nhập tên mới cho ảnh không có phần mở rộng + chọn “Lưu” + chờ tải

⇨ File ảnh sẽ có kích thước = Boundary của Flowchart, nền trắng, viền đen

7. Nhập Văn Bản Vào 1 Đối Tượng?

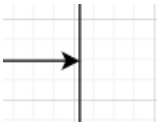
⇨ Double Click vào đối tượng + nhập văn bản + Click chỗ khác

8. Cấu Trúc Flowchart?

⇨ Luôn bao gồm 1 điểm bắt đầu và kết thúc, kí hiệu của chúng là hình chữ nhật bo viền, Start và End

⇨ Nếu có mũi tên vòng lại thì phải có đầu mũi tên

⇨ Minh họa



9. Chế Độ Kiểu Toán Học?

⇨ Để tắt bật chế độ này, vào Tab “Bổ sung” + Tick hoặc Untick “Sắp chữ kiểu toán học”

⇨ ở chế độ này, phần văn bản bên trong đối tượng có thể được nội suy, nếu nó bắt gặp một chuỗi con có dạng sau

```
$$<Văn Bản>$$
```

⇨ Thì nó xem chuỗi con này đang viết = Latex kiểu toán học, ví dụ để in ra mũi tên phải

```
foo$$A \rightarrow B$$bar
```

⇨ In ra

```
foo
A→B
bar
```

⇨ Phần toán học sẽ được hiển thị ở dòng riêng

10. Tạo Chữ Trên Mũi Tên Và Bám Theo Mũi Tên?

⇨ Double Click vào mũi tên + nhập nội dung + Click kéo thả để tạo Offset, khi này mũi tên thay đổi thì vị trí văn bản cũng thay đổi

11. Thay Đổi Ngôn Ngữ?

⇨ Vào Tab “Bổ sung” + chọn “Language” + chọn ngôn ngữ mong muốn + Click “OK” + Load lại trang

12. Vẽ Mũi Tên Đi Theo Đường Gấp Khúc Nào Đó?

⇨ Bước 1, kéo thả từ vị trí ban đầu theo đường thẳng đến điểm gấp khúc 1

⇨ Bước 2, phải chuột vào điểm này + chọn “Thêm điểm tham chiếu”

⇨ Bước 3, lặp lại bước 1, nhưng bắt đầu từ điểm này

⇨ Khi này nếu bạn di chuyển đầu mũi tên hoặc nguồn tạo ra mũi tên thì điểm tham chiếu cuối cùng không thay đổi vị trí, hình dạng đường đi thay đổi

13. Vẽ Đường Thay Cho Mũi Tên?

⇨ Ở phần bên trái + vào Tab “Chung” + Click biểu tượng “Line” để tạo ra 1 đoạn thẳng tại vị trí chính giữa trang + di chuyển đầu đoạn thẳng đến nguồn và đích, nó tự động dính vào

14. Vẽ Đường Gấp Khúc Với Đoạn Thẳng, Gộp 2 Tuyến Đường?

⇨ Thay vì tạo điểm tham chiếu, ta sử dụng Way Point

⇨ Để tạo 1 đối tượng Way Point

⇨ Double Click vào vị trí trống + chọn biểu tượng chấm đen

⇨ Bước 1, tạo Way Point tại các vị trí gấp khúc

⇨ Bước 2, tạo đoạn thẳng, nối 2 đầu của nó vào Way Point hoặc nguồn

⇨ Bước 3, di chuyển Way Point = di chuyển Vertex trong Blender

⇨ Từ Way Point, có thể xuất ra mũi tên

⇨ Để Click chọn Way Point, di chuột lên Way Point sao cho có 4 mũi tên mờ màu xanh chĩa ra + Click + bạn đã chọn thành công

⇨ Để ẩn chấm đen + chọn Way Point + tại phần bên phải cửa sổ Edit + vào Tab “Phong cách” + Untick “Dòng”, hoặc Tick nếu muốn hiện

Digital System – Hệ Thống Số:

1. Máy Tính Tính Các Hàm Sin, Cos, … Như Thế Nào?

⇨ Biến các hàm này thành chuỗi Taylor để tính, vì chuỗi Taylor chỉ bao gồm các phép tính căn bản như cộng trừ nhân chia lũy thừa

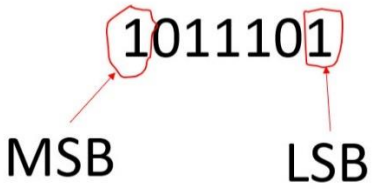
2. Cách Đọc Kí Hiệu Khoa Học?

⇨ Chữ e thì thay = 10 mũ

⇨ Ví dụ

$$8.6e+02 = 8.6 \times 10^2 = 860$$

3. LSB (Least Significant Bit) Và MSB (Most Significant Bit)?



4. Biểu Diễn Số Âm = Nhị Phân?

⇨ Cách 1, Sign Magnitude

⇨ Dùng Bit đầu làm dấu, quy ước 0 là + và 1 là –, các Bit sau làm độ lớn, ví dụ 1100 = –4

⇨ Cách 2, số bù 1

⇨ Dùng Bit đầu làm dấu, quy ước 0 là + và 1 là –, các Bit sau đảo rồi mới quy về độ lớn, ví dụ 1100 = 1(011) = –3

⇨ Cách 2, số bù 2

⇨ Bit đầu tiên sẽ được chọn để chỉ định dấu, quy ước 0 là + và 1 là –

⇨ Số + thì không có gì để nói, còn số – thì có chút thay đổi
⇨ Ví dụ
⇨ 1001 sẽ không phải là –1 mà là –7
⇨ Quy tắc 1
⇨ Ban đầu, tính bù 1, 1001 ứng với 0110
⇨ Tiếp theo, + 1 vào bù 1, 0110 + 1 = 0111
⇨ Cuối cùng, chuyển sang thập phân, 0111 thành 7, đặt dấu thành –7
⇨ Lí do phải làm như trên là vì khi ta cộng 2 số nhị phân có kiểu này thì sẽ ra kết quả đúng
⇨ Quy tắc 2
⇨ Cho Bit đầu tiên có giá trị âm

–8	4	2	1
1	0	0	1

⇨ 1001 = –8 + 1 = –7
⇨ Ví dụ
⇨ Ta muốn 2 + –5 phải = –3, ta có
⇨ 2 ứng với 0010, –5 ứng với 1011, 0010 + 1011 = 1101 ứng với –3
⇨ Lưu ý nếu + vào mà ra 5 Bit thì bỏ Bit đầu
⇨ Dễ thấy nếu ta dùng giá trị vượt quá tầm, như 10, –12, thì kết quả sẽ sai, vì các Bit đầu tiên sẽ bị loại bỏ do vượt quá 4 Bit
⇨ Bản chất, nếu kéo dài mãi số lượng Bit, thì nó nguyên dương sẽ có dạng như sau, ví dụ số 4
⇨ …00000100
⇨ Còn số nguyên âm, ví dụ số –5
⇨ …11111011

5. Biểu Diễn Số Thực = Nhị Phân?

⇨ Số thực thì mặc định có dấu
⇨ Phương pháp 1, sử dụng Fixed Point
⇨ Giả sử ta có 8 Bit, dùng 5 Bit đầu làm phần trước dấu phẩy, 3 Bit sau làm phần sau dấu phẩy
⇨ Minh họa

–16	8	4	2	1	0.5	0.25	0.125
0	0	1	0	1	0	1	1

⇨ 00101011 = 4 + 1 + 0.25 + 0.125 = 5.375
⇨ 10110100 = –16 + 4 + 2 + 0.5 = –9.5
⇨ Để + 2 cái lại, thì cứ + như thường, bỏ các Bit đầu tiên nếu vượt quá 8 Bit
⇨ 00101011 + 10110100 = 11011111 = –4.125
⇨ Để chuyển ngược lại từ số thực sang nhị phân, ví dụ -12.3

–16 < –12.3, đánh 1 vào ô –16
–16 + 8 = –8 > –12.3, đánh 0 vào ô 8
–16 + 4 = –12 > –12.3, đánh 0 vào ô 4
–16 + 2 = –14 < –12.3, đánh 1 vào ô 2
–14 + 1 = –13 < –12.3, đánh 1 vào ô 1
–13 + 0.5 = –12.5 < –12.3, đánh 1 vào ô 0.5
–12.5 + 0.25 = –12.25 > –12.3, đánh 0 vào ô 0.25
–12.5 + 0.125 = –12.375 < –12.3, đánh 1 vào ô 0.125

⇨ Như vậy –12.3 được biểu diễn là 10011101 = –12.375, có sai số
⇨ Phương pháp 2, dùng Floating Point (IEEE 754)

- ⇨ Bản chất của cách này là dựa theo việc ta có thể biểu diễn số dưới dạng a x 10^b
- ⇨ Giả sử ta có 32 Bit, Bit đầu làm dấu, 8 Bit sau làm Exponent, 23 Bit cuối làm Mantissa
- ⇨ Giả sử ta có số 01000010101001100110011001100110, biểu diễn lại thành 0|10000101|01001100110011001100110
- ⇨ Quy đổi phần Exponent
- ⇨ 10000101 = 133, đặt Bit đầu tiên = 0, còn lại = 1, 01111111 = 127, lấy 133 – 127 = 6, vậy Exponent = 6
- ⇨ Quy đổi phần Mantissa
- ⇨ Đặt 1. phía trước được 1.01001100110011001100110
- ⇨ Dịch dấu chấm sang phải Exponent = 6 đơn vị, được 1010011.00110011001100110
- ⇨ Trường hợp Exponent số âm, thì dịch sang trái, ví dụ Exponent = –5 thì dịch sang trái 5 đơn vị, lưu ý phải mở rộng phần Mantissa sang trái = cách thêm Bit 0, không cắt đi phần bên phải
- ⇨ Quy đổi sang số thực như khi dùng Fixed Point
- ⇨ 1010011.00110011001100110 = 83.1999969482…
- ⇨ Bit làm dấu có giá trị = 0, tương ứng số dương, vậy kết quả cuối cùng là 83.1999969482…
- ⇨ Nếu Bit làm dấu có giá trị = 1, thì đặt dấu âm phía trước
- ⇨ Để chuyển từ số thực sang nhị phân
- ⇨ Giả sử ta có số 83.2, biểu diễn nó dưới dạng nhị phân mở rộng như khi dùng Fixed Point
- ⇨ 83.2 = …0001010011.0011001100110011001…
- ⇨ Di chuyển dấu chấm sang trái k = 6 đơn vị sao cho nó ngay đứng bên phải Bit 1 ngoài cùng bên trái, sau đó cắt bỏ các Bit bên trái dấu chấm và các Bit bên phải cùng sao cho số Bit còn lại = 23 ta được
- ⇨ 01001100110011001100110
- ⇨ Đây chính là phần Mantissa
- ⇨ Chuyển k + 127 sang nhị phân, 6 + 127 = 133 = 10000101, đây chính là phần Exponent
- ⇨ Do 83.2 dương nên Bit dấu = 0
- ⇨ Vậy cuối cùng ta được

83.2 = 01000010101001100110011001100110

- ⇨ Để + 2 Floating Point
- ⇨ Giả sử ta có a = 83.2 = 01000010101001100110011001100110 và b = 4.53 = 01000000100100001111010111000011
- ⇨ Đầu tiên, xác định Exponent của 2 số
- ⇨ Exponent của a = 10000101 = 6, của b = 10000001 = 2
- ⇨ Phải cân bằng Exponent của 2 bên lên giá trị lớn hơn, ở đây là 6, do đó Exponent của b phải từ 2 lên 6, đặt Exponent cuối = 6
- ⇨ Mantissa của b = 00100001111010111000011, đặt 1 . vào trước, sau đó dịch dấu chấm sang trái 6 – 2 = 4 đơn vị, cắt bỏ các Bit bên phải cùng sao cho số Bit bên phải dấu chấm = 23, ta được
- ⇨ 0.00010010000111101011100
- ⇨ Đặt 1. Vào trước Mantissa của a, sau đó + 2 cái Mantissa lại với nhau

1.01001100110011001100110 +

0.000100100000111101011100 = 1.010111101110101111000010

⇨ Nếu 2 Bit đầu của 2 số hạng đều = 1, khi đó, 1 + 1 = 10, dịch dấu chấm sang trái 1 đơn vị, bỏ Bit phải cùng, và Exponent cuối + thêm 1, ví dụ từ 6 lên 7

⇨ Bỏ 1. được 01011110111010111000010, đây chính là phần Mantissa của kết quả

⇨ Quy đổi Exponent cuối, 6 = 10000101

⇨ Cuối cùng, ghép tất cả lại, ta được

83.2 + 4.53 = 01000010101011110111010111000010 = 87.72999572…

6. Nhân Chia 2 Số Nguyên Hoặc 2 Fixed Point Dưới Dạng Nhị Phân?

⇨ Y chang các bước khi bạn làm với thập phân, và có cắt bỏ bớt các Bit bên phải

7. Chuyển Nhị Phân Sang Thập Lục Phân?

⇨ Chia nhị phân thành từng nhóm 4 rồi quy đổi từng nhóm

⇨ Ví dụ

⇨ 101011101 ứng với 1|0101|1101 ứng với 15D

⇨ Tương tự khi chuyển nhị phân sang bát phân, chia thành từng nhóm 3

8. Bảng Mã BCD (NBCD)?

⇨ Thay vì chia 2 lấy dư các kiểu để chuyển thập phân sang nhị phân, ta dùng bảng mã BCD, thế mỗi số thập phân = số nhị phân 4 Bit tương ứng

⇨ Ví dụ

⇨ Số 369, 3 = 0011, 6 = 0110, 9 = 1001, nên 369 = 0011 0110 1001

⇨ Để cộng các số trong dạng BCD, ta cộng nhị phân như bình thường, nhưng nếu giá trị vượt quá 9 thì + thêm 6

⇨ Ví dụ

⇨ 369 + 457 = 826

⇨ Để tính phép tính trên, đổi 369 = 0011 0110 1001, 457 = 0100 0101 0111

⇨ Cộng từng cặp 4 Bit, 1001 + 0111 = 10000 = 16, nên + thêm 6 thành 10000 + 0110 = 10110 = 0110 nhớ 1, 0110 chuyển thành 6

⇨ 0110 + 0101 + 1 = 1100 = 12, nên + thêm 6 thành 1100 + 0110 = 10010 = 0010 nhớ 1, 0010 chuyển thành 4

⇨ 0011 + 0100 + 1 = 1000 = 8, không + thêm 6

⇨ Ghép tất cả lại, được 826

9. Chuyển Nhị Phân Sang BCD Bằng Thuật Toán Double Dabble?

⇨ Bước 1, kẻ dãy ô sau, mỗi chùm 4 Bit sẽ làm 1 chữ số BCD, số chùm 4 Bit = số Bit nhị phân / 3 làm tròn lên, ví dụ số nhị phân = 987 = 1111011011 thì số chùm 4 Bit = 10 / 3 làm tròn lên = 4, khởi tạo giá trị ban đầu = 0

0000	0000	0000	0000	1111011011

⇨ Bước 2, lặp qua tất cả chùm 4 Bit, nếu giá trị của nó từ 5 trở lên thì + thêm 3

⇨ Bước 3, dịch trái tất cả 1 bước

⇨ Bước 4, nếu ô màu vàng không trống thì trở lại bước 2, nếu trống thì kết thúc thuật toán, như vậy số lần lặp = số Bit nhị phân

⇨ Với ví dụ trên, sau 3 lần lặp, ta được

0000	0000	0000	0111	1011011

⇨ Lần lặp thứ 4, ô thứ 4 sẽ + 3 trước khi dịch

0000	0000	0000	1010	1011011
0000	0000	0001	0101	011011

⇨ Tương tự, lần lặp thứ 5

0000	0000	0001	1000	011011

0000	0000	0011	0000	11011

⇨ Lần lặp thứ 6

0000	0000	0110	0001	1011

⇨ Lần lặp thứ 7

0000	0000	1001	0001	1011
0000	0001	0010	0011	011

⇨ Lần lặp thứ 8

0000	0010	0100	0110	11

⇨ Lần lặp thứ 9

0000	0010	0100	1001	11
0000	0100	1001	0011	1

⇨ Lần lặp thứ 10

0000	0100	1100	0011	1
0000	1001	1000	0111	

⇨ Sau 10 lần lặp, ô vàng trống, như vậy số BCD = 0000 1001 1000 0111 = 0987

10. Bảng Mã Excess 3?

⇨ Y chang BCD, nhưng cộng thêm 3 vào dạng nhị phân

⇨ Ví dụ

⇨ Số 369, 3 = 0011 + 3 = 0110, 6 = 0110 + 3 = 1001, 9 = 1001 + 3 = 1100, nên 369 = 0110 1001 1100

⇨ Để trả về số bù 9, ta chỉ việc phủ định các Bit, ví dụ 3 = 0110, ta muốn 6, đảo các Bit thành 1001 = 6

⇨ Để cộng các số trong dạng Excess 3, ta cộng nhị phân như bình thường từng bộ 4 Bit, sau đó bộ nào mà không dư ra 1 Bit ở đầu thì – 6, bộ bên trái cùng thì luôn – 6, sau đó Bit thừa của bộ 4 Bit này sẽ cộng cho bộ 4 Bit bên trái, cuối cùng, đổi tất cả về số nguyên tương ứng rồi ghép lại

⇨ Ví dụ

⇨ 865 + 454 = 1319

⇨ Để tính phép tính trên, đổi 869 = 1011 1001 1000, 459 = 0111 1000 0111

⇨ Cộng từng bộ 4 Bit được 10010 10001 1111

⇨ Bộ cuối không dư 1 Bit, – 6, bộ đầu luôn – 6, được 1100 10001 1001

⇨ Chuyển Bit dư sang trái, được 1101 0001 1001

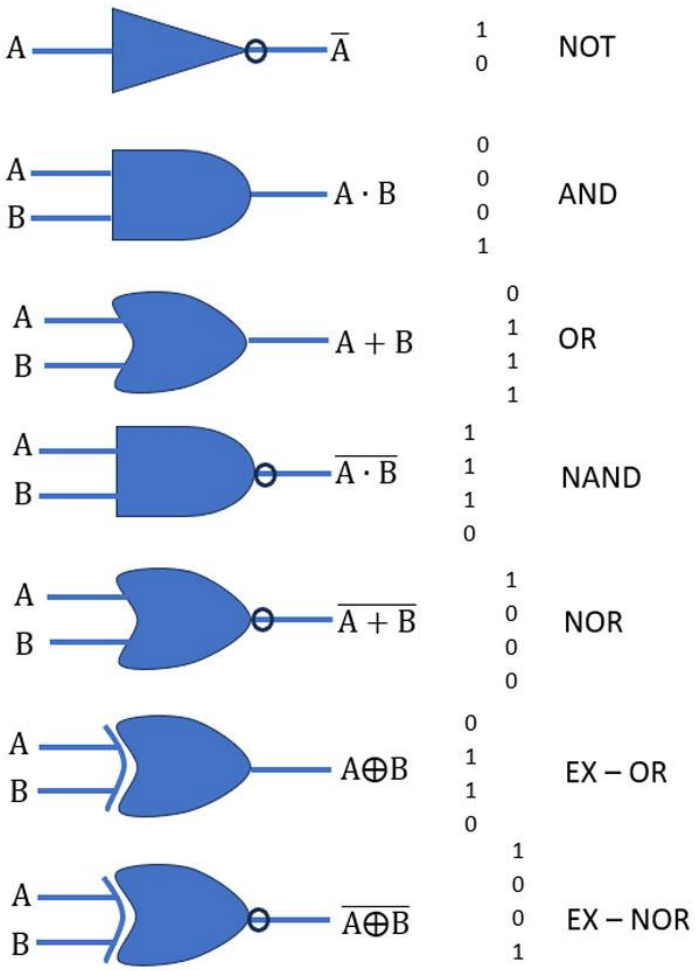
⇨ Đổi về số nguyên tương ứng, được 13 1 9, ghép lại thành 1319

11. Bảng Mã Kí Tự (Alphanumeric Code)?

⇨ Ví dụ như bảng mã ASCII dùng 7 Bit, “A” là 1000001, “a” là 1100001, “0” là 0110000

12. Tất Cả Cá Cổng Logic?

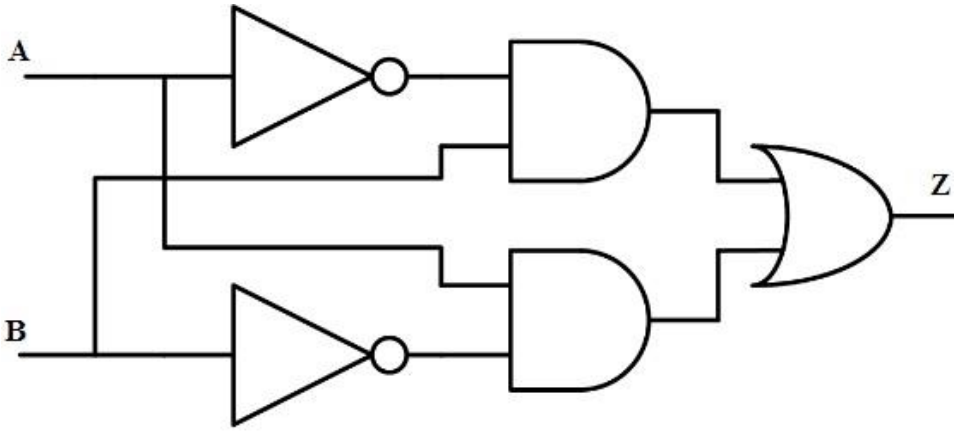
⇨ Minh họa



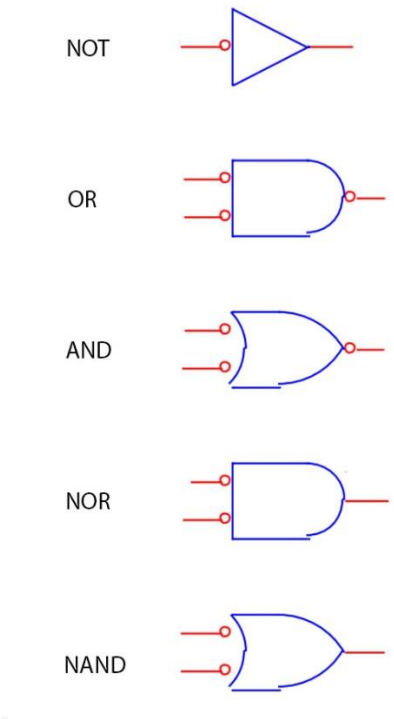
⇨ Tất cả các phép toán trên đều có tính giao hoán
⇨ Có tính chất phân phối giữa nhân và cộng, hay AND và OR
⇨ Chỉ có AND, OR, EX – OR, EX – NOR có tính kết hợp
⇨ Lưu ý cổng NAND, NOR, EX – NOR có số Input > 2, = cổng AND, OR, EX – OR tương ứng + cổng NOT

13. Các Kí Hiệu Cổng Logic Thay Thế?

⇨ Giả sử 1 thằng nào đó quăng cho ta 1 mạch Logic và bắt ta lập bảng thực trị, thì lâu vãi lồn, để làm nhanh, dùng phương pháp suy luận ngược
⇨ Ví dụ cho mạch sau

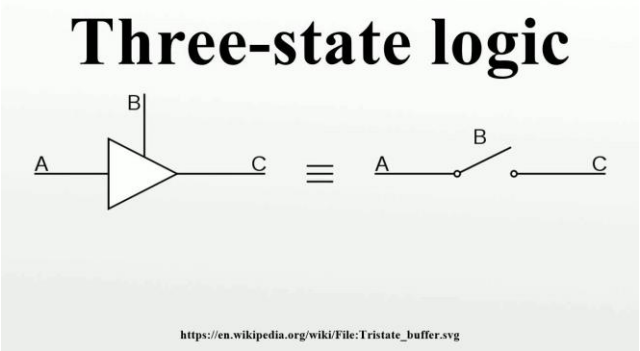


- ⇨ Bắt đầu suy luận, để Z = 0 thì 2 đầu vào cổng OR đằng trước phải = 0, để đầu vào bên trên = 0 thì ít nhất 1 trong 2 chân của cổng AND đằng trước phải = 0, …
- ⇨ Cứ thế ta suy luận những trường hợp nào của A và B sẽ cho ra Z = 0 thì chỉ cần điền 0 vào những tổ hợp tương ứng
- ⇨ Để thuận tiện hơn cho việc suy luận ngược, ta sử dụng kí hiệu thay thế của các cổng, quy tắc OR thì đổi sang hình AND và ngược lại



- ⇨ Ví dụ cách đọc các kí hiệu này
- ⇨ Với kí hiệu OR, ta dùng Shape AND, 2 đầu vào tròn, đầu ra tròn, tròn thì quy về 0, nghĩa là để đầu ra = 0 thì cả 2 đầu vào phải = 0
- ⇨ Với kí hiệu NAND, ta dùng Shape OR, 2 đầu vào tròn, đầu ra không có gì, quy ước không có gì thì là 1, nghĩa là để đầu ra = 1 thì ít nhất 1 trong 2 đầu vào = 0
- ⇨ Tương tự với các kí hiệu khác
- ⇨ Các kí hiệu hiệu này = Bubbled + tên thường, ví dụ AND = Bubbled NOR, NOR = Bubbled AND

14. Cổng Logic 3 trạng thái (Three State)?

- ⇨ 

- ⇨ Khi B = 1, thì tương đương A nối thẳng vào C, A bao nhiêu C bây nhiêu

⇨ Khi B = 0, tương đương giữa A và C mạch hở, C có giá trị ngẫu nhiên (High Impedance)

15. Tích Cực Mức Cao (Active High) Và Tích Cực Mức Thấp (Active Low)?

⇨ 1 ngõ nhập hoặc xuất gọi là tích cực mức cao thì tên của nó, hay chức năng của nó đạt được khi nó = 1, ví dụ ngõ tên "Bật đèn sáng", nếu nó tích cực mức cao thì khi nó = 1 đèn phải sáng và = 0 thì đèn phải tối

⇨ 1 ngõ nhập hoặc xuất gọi là tích cực mức thấp thì tên của nó, hay chức năng của nó đạt được khi nó = 0, ví dụ ngõ tên "Bật đèn sáng", nếu nó tích cực mức thấp thì khi nó = 0 đèn phải sáng và = 1 thì đèn phải tối, ngõ nhập loại này sẽ có biểu tượng hình tròn ở đầu

16. Bit Chẵn Lẻ (Parity Bit)?

⇨ Giả sử ta muốn truyền chuỗi Bit 10101110 cho 1 người, nhưng vì lí do nào đó mà nó trở thành 10101111, người nhận thì đéo biết có lỗi hay không và kết quả dịch sai

⇨ Để cho phép người nhận biết rằng chuỗi mình nhận có lỗi hay không, ta truyền thêm cho họ 1 Parity Bit

⇨ Nếu là Even Parity Bit, thì giá trị của nó = 0 khi số Bit 1 là số chẵn, và = 1 nếu lẻ

⇨ Nếu là Odd Parity Bit thì ngược lại

⇨ Người nhận = cách đếm số Bit 1 và Check loại Parity Bit là chẵn hay lẻ là biết có lỗi hay không

⇨ Parity Bit đặt ở đầu, ví dụ 10101111 + Parity Bit chẵn = 010101111

17. Các Công Thức Logic Quan Trọng?

$$A = A \cdot A = A \cdot 1 = A + A = A + 0 = (A + B) \cdot (A + \bar{B})$$
$$1 = A + 1 = A + \bar{A}$$
$$0 = A \cdot 0 = A \cdot \bar{A}$$
$$A + \bar{A} \cdot B = A + B$$
$$A \cdot B + \bar{B} \cdot C + C \cdot A = A \cdot B + \bar{B} \cdot C$$
$$(A + B) \cdot (A + C) = A + B \cdot C$$
$$\overline{A + B + C + \cdots} = \bar{A} \cdot \bar{B} \cdot \bar{C} \cdot \ldots$$
$$\overline{A \cdot B \cdot C \cdot \ldots} = \bar{A} + \bar{B} + \bar{C} + \cdots$$
$$A \oplus B = A \cdot \bar{B} + \bar{A} \cdot B = (A + B) \cdot (\bar{A} + \bar{B})$$
$$\overline{A \oplus B} = \bar{A} \oplus B = \bar{A} \cdot \bar{B} + A \cdot B = (A + \bar{B}) \cdot (\bar{A} + B)$$

⇨ Công thức chuyển đổi sang Full NAND hoặc NOR

$$\bar{A} = \overline{A \cdot A} = \overline{A + A}$$
$$A \cdot B \cdot C \cdot \ldots = \overline{\overline{A \cdot B \cdot C \cdot \ldots} \cdot \overline{A \cdot B \cdot C \cdot \ldots}} = \overline{\overline{A + A} + \overline{B + B} + \overline{C + C} + \cdots}$$
$$A + B + C + \cdots = \overline{\overline{A \cdot A} \cdot \overline{B \cdot B} \cdot \overline{C \cdot C} \cdot \ldots} = \overline{\overline{A + B + C + \cdots} + \overline{A + B + C + \cdots}}$$

18. Cách Chuyển 1 Mạch Chỉ Có Cổng AND, OR, NOT 2 Ngõ Nhập Sang Mạch Full NAND Hoặc Full NOR Nhanh Nhất?

⇨ Để Full NAND

⇨ Bước 1, chuyển tất cả cổng AND thành NAND + NOT

⇨ Bước 2, chèn cổng NOT vào trước ngõ nhập của tất cả cổng OR

⇨ Bước 3, thay tất cả cổng OR thành NAND

⇨ Bước 4, loại bỏ các cổng NOT dư

⇨ Bước 5, thay tất cả cổng NOT thành NAND

⇨ Để Full OR

⇨ Y chang như trên, thay chữ "OR" thành "AND" và ngược lại

⇨ Ví dụ, chuyển công thức về Full NAND, dấu "⊗" là NAND, bình phương nghĩa là NAND với chính nó

AB' + ACD' + B'D + A'C'D = (((AB') + ((AC)D')) + (B'D)) + ((A'C')D) =
(((A⊗B')' + ((A⊗C)'⊗D')') + (B'⊗D)') + ((A'⊗C')'⊗D)' =
(((A⊗B') + ((A⊗C)'⊗D'))' + (B'⊗D))' + ((A'⊗C')'⊗D) =
(((A⊗B')⊗((A⊗C)'⊗D'))'⊗(B'⊗D))'⊗((A'⊗C')'⊗D) =
$(((A\otimes B^2)\otimes((A\otimes C)^2\otimes D^2))^2\otimes(B^2\otimes D))^2\otimes((A^2\otimes C^2)^2\otimes D)$

⇨ Sơ đồ mạch Full NAND hoặc NOR làm theo cách này chưa phải tối giản nhất

⇨ Xem sơ đồ mạch Full NAND hoặc NOR tối giản nhất của 1 số biểu thức tại "Exercise\he_thong_so\latch_and_flip_flop_and_ic.circ"

19. Cách Lập Công Thức Cho Trước 1 Bảng Thực Trị?

⇨ Giả sử ta có bảng sau

A	B	C	X = f(A, B, C)
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

⇨ Bây giờ ta muốn tìm hàm f thỏa mãn bảng trên, để làm được điều này, có 2 cách

⇨ Cách 1, sử dụng dạng chuẩn tuyển (Minterm, DNF, Disjunctive Normal Form)

$$f(A,B,C) = \begin{matrix}\bar{A}\cdot\bar{B}\cdot\bar{C}\cdot 1 + \bar{A}\cdot\bar{B}\cdot C\cdot 0 + \bar{A}\cdot B\cdot\bar{C}\cdot 0 + \bar{A}\cdot B\cdot C\cdot 0 + \\ A\cdot\bar{B}\cdot\bar{C}\cdot 1 + A\cdot\bar{B}\cdot C\cdot 1 + A\cdot B\cdot\bar{C}\cdot 0 + A\cdot B\cdot C\cdot 1\end{matrix} =$$
$$\bar{A}\cdot\bar{B}\cdot\bar{C}\cdot 1 + A\cdot\bar{B}\cdot\bar{C}\cdot 1 + A\cdot\bar{B}\cdot C\cdot 1 + A\cdot B\cdot C\cdot 1 =$$
$$\bar{A}\cdot\bar{B}\cdot\bar{C} + A\cdot\bar{B}\cdot\bar{C} + A\cdot\bar{B}\cdot C + A\cdot B\cdot C = \bar{A}\cdot\bar{B}\cdot\bar{C} + A\cdot\bar{B} + A\cdot B\cdot C =$$
$$\bar{A}\cdot\bar{B}\cdot\bar{C} + A\cdot\bar{B} + A\cdot C = \bar{B}\cdot(A+\bar{C}) + A\cdot C$$

⇨ Tóm lại, dạng chuẩn tuyển thực chất là sử dụng tổng các tích, để ý thấy, khi A = B = C = 0 thì tích đầu tiên mới có tác dụng, còn các tích còn lại đều = 0, tương tự khi A = B = 0, C = 1 thì chỉ có tích thứ 2 có tác dụng, còn lại đều = 0, do đó ta nhân mỗi tích thêm giá trị tương ứng của hàm f tại tổ hợp giá trị đó là ra

⇨ Cách 2, sử dụng dạng chuẩn hội (Maxterm, CNF, Conjunctive Normal Form)

$$f(A,B,C) =$$
$$\begin{matrix}(A+B+C+1)\cdot(A+B+\bar{C}+0)\cdot(A+\bar{B}+C+0)\cdot(A+\bar{B}+\bar{C}+0)\cdot \\ (\bar{A}+B+C+1)\cdot(\bar{A}+B+\bar{C}+1)\cdot\ (\bar{A}+\bar{B}+C+0)\cdot(\bar{A}+\bar{B}+\bar{C}+1)\end{matrix} =$$
$$(A+B+\bar{C}+0)\cdot(A+\bar{B}+C+0)\cdot(A+\bar{B}+\bar{C}+0)\cdot(\bar{A}+\bar{B}+C+0) =$$
$$(A+B+\bar{C})\cdot(A+\bar{B}+C)\cdot(A+\bar{B}+\bar{C})\cdot(\bar{A}+\bar{B}+C) =$$
$$(A+B\cdot C+\bar{B}\cdot\bar{C})\cdot(\bar{B}+A\cdot C+\bar{A}\cdot\bar{C}) =$$
$$A\cdot\bar{B}+A\cdot C+A\cdot B\cdot C+\bar{B}\cdot\bar{C}+\bar{A}\cdot\bar{B}\cdot\bar{C} = A\cdot C+\bar{B}\cdot\bar{C}+A\cdot\bar{B} =$$
$$\bar{B}\cdot(A+\bar{C})+A\cdot C$$

⇨ Tóm lại, dạng chuẩn hội thực chất là sử dụng tích các tổng, để ý thấy, khi A = B = C = 0 thì tổng đầu tiên mới có tác dụng, còn các tổng còn lại đều = 1, tương tự khi A = B = 0, C = 1 thì chỉ có tích thứ 2 có tác dụng, còn lại đều = 1, do đó ta cộng mỗi tổng thêm giá trị tương ứng của hàm f tại tổ hợp giá trị đó là ra

⇨ Tổng quát, dù hàm f có bao nhiêu biến đi nữa thì ta vẫn dễ dàng lập được công thức cho bảng thực trị bất kì

20. Tổ Hợp Đéo Quan Tâm (Don't Care)?

⇨ Có 1 cách khác để biểu diễn hàm Logic nhiều biến

⇨ Trường hợp 1, viết theo dạng chuẩn tuyển

$$f(A,B,C,\dots) = \sum m(a,b,c,\dots) + d(x,y,z,\dots)$$

⇨ Có nghĩa là tại tổ hợp biến cho dãy nhị phân, mà khi đổi ra số nguyên, thì thành a, b, c, …, mà tổ hợp biến này lại trả về 1 khi đi qua f

⇨ Tương tự với x, y, z, …, nhưng giá trị trả về của f ứng với tổ hợp biến cho ra x, y, z, … mình đéo quan tâm, nghĩa là nó trả về 0 hay 1 gì đều được

⇨ Ví dụ có bảng thực trị, chỗ đánh dấu x là giá trị trả về mình đéo quan tâm

A	B	C	X = f(A, B, C)
0	0	0	1
0	0	1	0
0	1	0	x
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	x
1	1	1	1

⇨ Ta có

$$f(A,B,C) = \sum m(0,4,5,7) + d(2,6)$$

⇨ Trường hợp 2, viết theo dạng chuẩn hội

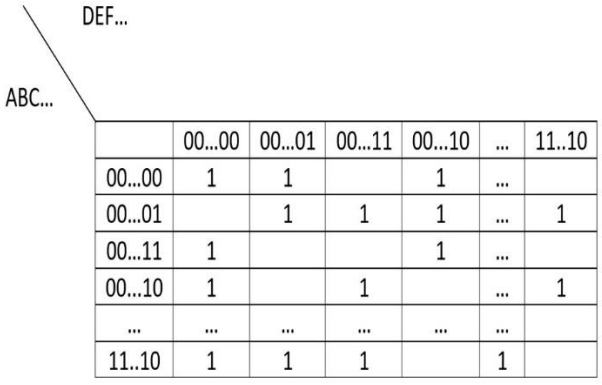
$$f(A,B,C,\dots) = \prod M(a,b,c,\dots) \cdot D(x,y,z,\dots)$$

⇨ Đéo khác gì trường hợp 1, chỉ có điều a, b, c, … là những chỗ f trả về 0

21. Rút Gọn Hàm Nhanh Hơn Với Biểu Đồ Karnaugh?

⇨ Bước 1, vẽ biểu đồ

⇨ Biểu đồ có dạng sau



	00...00	00...01	00...11	00...10	...	11..10
00...00	1	1		1	...	
00...01		1	1	1	...	1
00...11	1			1	...	
00...10	1		1		...	1
...	...	...	...	...	...	
11..10	1	1	1		1	

⇨ ABC… là nửa các biến đầu theo đúng thứ tự trong bảng thực trị

⇨ DEF… là nửa các biến sau theo đúng thứ tự trong bảng thực trị

⇨ Nếu số biến lẻ thì cho nửa các biến sau nhiều hơn

⇨ Hàng trên cùng, tương ứng tổ hợp nhị phân của DEF…, mỗi tổ hợp phải khác nhau đúng 1 Bit,

⇨ Để làm điều này, dùng mẹo sau

⇨ Với 1 Bit, ta có thứ tự 0, 1

- ⇨ Với 2 Bit, thêm đảo của 1 Bit, được 0, 1, 1, 0, sau đó thêm 0 vào nửa trước, 1 vào nửa sau, được 00, 01, 11, 10
- ⇨ Với 3 Bit, thêm đảo của 2 Bit, được 00, 01, 11, 10, 10, 11, 01, 00, sau đó thêm 0 vào nửa trước, 1 vào nửa sau, được 000, 001, 011, 010, 110, 111, 101, 100
- ⇨ Vân vân, đây còn gọi là bảng mã Gray, 0 ứng với 0, 1 ứng với 1, 11 ứng với 2, 10 ứng với 3, 110 ứng với 4, …, bảng mã này có lợi thế là ví dụ nếu ta muốn tăng 7 lên 8 thì chỉ cần đảo 1 Bit, thay vì đảo có khi hết toàn bộ Bit, nhưng lại không thể cộng
- ⇨ Tương tự với cột ngoài cùng bên trái
- ⇨ Dễ thấy 1 ô trong biểu đồ này tương ứng 1 hàng trong bảng thực trị, điền giá trị f trả về tại hàng đó vào nếu là 1
- ⇨ Bước 2, khoanh vùng
- ⇨ Lặp qua tất cả các ô, từ phải qua trái, từ trên xuống dưới
- ⇨ Nếu gặp ô 1, dừng lại để khoanh vùng, quy tắc vùng phải là hình chữ nhật có số ô = lũy thừa của 2, chỉ chứa 1 và Don't Care, có thể vượt biên như rắn săn mồi, xác định kích thước của vùng lớn nhất có thể mà ô này thuộc về, gọi là k, khoanh tất cả các vùng có kích thước k mà ô này thuộc về, đánh số thứ tứ tăng thêm 1 cho đỡ rối, chỉ khoanh vùng khi đây là vùng mới
- ⇨ Sau khi hoàn thành, bắt đầu lặp qua các vùng, từ số thứ tự lớn nhất đến 1, vùng nào mà không có ô 1 nào chỉ thuộc về nó thì xóa
- ⇨ Bước 3, tối giản hàm
- ⇨ Tìm số hạng của tất cả các vùng rồi cộng chúng lại
- ⇨ Cách tìm số hạng tương ứng của 1 vùng
- ⇨ Viết các biểu diễn nhị phân của từng ô từ trên xuống dưới, ví dụ ta có vùng 4 ô (màu vàng), hàm 4 biến theo thứ tự A, B, C, D

	00	01	11	10
00				
01				
11				
10				

- ⇨ Ta viết

1101
1111
1001
1011

- ⇨ Sau đó, chỉ giữ lại các cột có giá trị thuần 1 hoặc 0, ở đây là cột đầu và cột cuối, do đều là 1 nên ta viết AD, nếu thuần 0 thì thêm gạch lên đầu
- ⇨ Để tìm dạng chuẩn hội tối giản thì cũng y chang đéo khác gì, chỉ thay vì chọn ô 1 thì chọn ô 0, lưu ý bây giờ A là 0, A' là 1, nhân thành cộng, cộng thành nhân
- ⇨ Ví dụ, rút gọn hàm sau

$f(A,B,C,D) = \sum m(0,4,5,6,7,8,9,10,11,13,14,15)$

- ⇨ Biểu đồ

	00	01	11	10
00	1			
01	1	1	1	1
11		1	1	1

10	1	1	1	1

⇨ Khoanh vùng

	00	01	11	10
00	1 12			
01	1 13	1 34	1 345	1 35
11		1 46	1 4567	1 57
10	1 2	1 6	1 67	1 7

⇨ Ví dụ 46 nghĩa là ô này thuộc về vùng 4 và vùng 6

⇨ Xóa khoanh vùng, đi từ 7 đến 1

	00	01	11	10
00	1 2			
01	1 3	1 3	1 3	1 3
11		1 6	1 67	1 57
10	1 2	1 6	1 67	1 7

⇨ Tính số hạng các vùng còn tồn tại

⇨ Vùng 2 = B'C'D'

⇨ Vùng 3 = A'B

⇨ Vùng 6 = AD

⇨ Vùng 7 = AC

⇨ Tổng các số hạng lại, ta được

$$f(A,B,C,D) = \bar{A}\cdot B + A\cdot D + A\cdot C + \bar{B}\cdot\bar{C}\cdot\bar{D}$$

⇨ Đây chính là dạng chuẩn tuyển tối giản của f

22. Mạch Full EX – OR Hoặc EX – NOR?

⇨ Giả sử ta đang lập 1 biểu đồ Karnaugh, và ta nhận thấy rằng tất cả các ô 1 trong biểu đồ xếp theo kiểu bàn cờ vua

⇨ Giả sử xét hàm 4 biến, thì biểu đồ có kiểu sau

	00	01	11	10
00		1		1
01	1		1	
11		1		1
10	1		1	

	00	01	11	10
00	1		1	
01		1		
11	1		1	
10		1		1

⇨ Chọn 1 hình chữ nhật bất kì có số ô là lũy thừa của 2, giả sử hình sau

	00	01	11	10
00		1		1
01	1		1	
11		1		1
10	1		1	

⇨ Để rút gọn vùng này, viết các biểu diễn nhị phân của từng ô từ trên xuống dưới

0100
0101
0111
0110

1100
1101
1111
1110

⇨ Xác định các cột có giá trị thuần 1 hoặc 0, ở đây có cột thứ 2, tích các cột này lại, lưu ý phủ định nếu thuần 0

⇨ Các cột còn lại cho EX – OR với nhau nếu ô đầu tiên của hình chữ nhật = 0, còn không thì EX – NOR với nhau

⇨ Tích 2 cái lại, ở ví dụ này, ta được

$$B \cdot \left(\overline{A \oplus C \oplus D}\right)$$

⇨ Ví dụ khác, rút gọn nguyên bảng sau

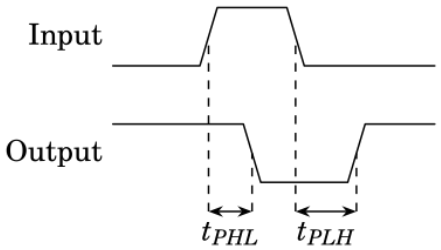
	00	01	11	10
00		1		1
01	1		1	
11		1		1
10	1		1	

⇨ Ta được

$$A \oplus B \oplus C \oplus D$$

23. Độ Trễ Lan Truyền (Propagation Delay) Của 1 Cổng?

⇨ Xét cổng NOT, đồ thị Input Output như sau



⇨ Độ trễ lan truyền là khoảng thời gian giữa thời điểm Input thay đổi được 50% giá trị tới thời điểm Output thay đổi được 50% giá trị, nghĩa Output chưa thay đổi ngay khi Input thay đổi

⇨ Độ trễ lên (Rise Delay) = độ trễ lan truyền khi Output thay đổi từ 0 lên 1

⇨ Độ trễ xuống (Fall Delay) = độ trễ lan truyền khi Output thay đổi từ 1 xuống 0

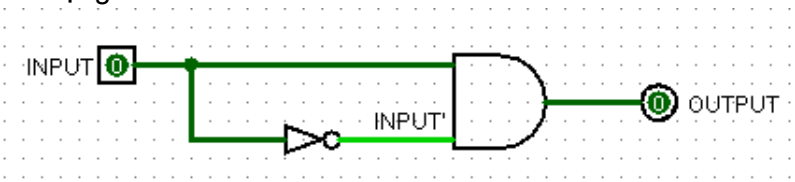
⇨ Thời gian lên (Rising Time) của 1 tín hiệu là khoảng thời gian để nó chuyển 0.1 lên 0.9

⇨ Thời gian xuống (Falling Time) của 1 tín hiệu là khoảng thời gian để nó chuyển từ 0.9 xuống 0.1

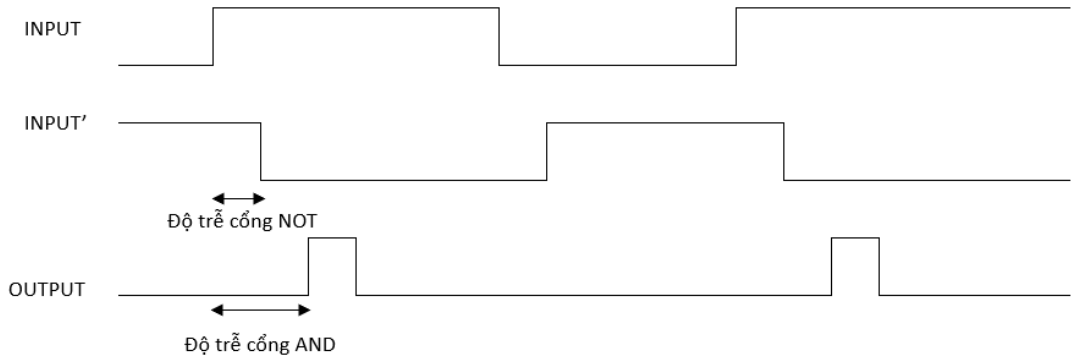
⇨ Độ rộng xung sẽ tính từ thời điểm tín hiệu đạt 50%

24. Edge Detector?

⇨ Có dạng như sau



⇨ Do độ trễ lan truyền, ta có sơ đồ thời gian



⇨ Như vậy, chỗ nào của tín hiệu Input chuyển từ 0 lên 1 sẽ được nhận dạng, tạo xung gai

⇨ Nếu muốn nhận dạng những cạnh từ 1 xuống 0, thì thay cổng AND thành NOR

25. Latch?

Xem cấu trúc các Latch tại "Exercise\he_thong_so\latch_and_flip_flop_and_ic.circ"

⇨ SR Latch

⇨ Bảng thực trị

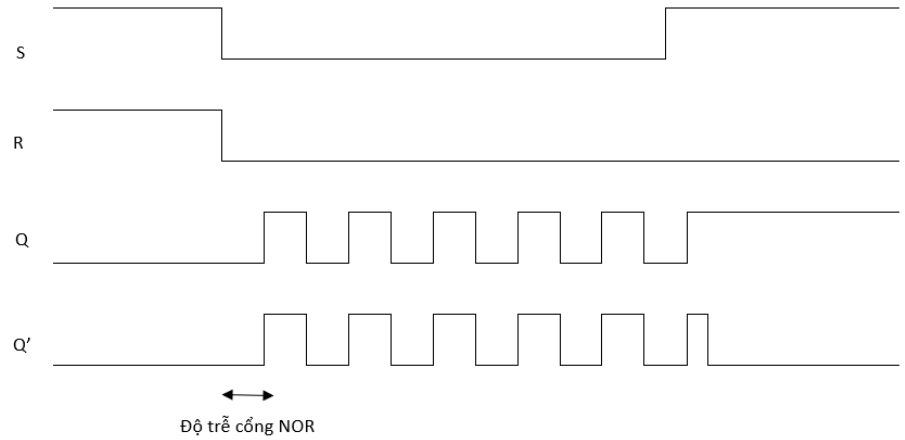
		Tích cực mức cao		Tích cực mức thấp	
S	R	Q	Q'	Q	Q'
0	0	0	1	1	1
		1	0		
0	1	0	1	1	0
1	0	1	0	0	1
1	1	0	0	0	1
				1	0

⇨ Xét tích cực mức cao

⇨ Để tính Q và Q' thì lợi dụng độ trễ lan truyền giữa các cổng, tính từng cổng một

⇨ Nếu mạch đang ở trạng thái Q = 0, Q' = 1, hoặc Q = 1, Q' = 0 rồi thì khi bạn thay đổi S hoặc R sao cho S = R = 0, thì Q và Q' không đổi

⇨ Nếu cả S và R đều từ 1 xuống 0 cùng lúc thì Q và Q' sẽ dao động, ta có sơ đồ thời gian



⇨ Gated SR Latch

⇨ E được gọi là ngõ nhập đồng bộ

⇨ Bảng thực trị khi E = 1

		Loại	Loại

		NOR		NAND	
S	R	Q	Q’	Q	Q’
0	0	0	1	0	1
		1	0	1	0
0	1	0	1	0	1
1	0	1	0	1	0
1	1	0	0	1	1

⇨ Khi E = 0 thì như E = 1 và S = R = 0

⇨ Gated D Latch

⇨ Bảng thực trị

E	D	Q	Q’
0	0	0	1
		1	0
0	1	0	1
		1	0
1	0	0	1
1	1	1	0

⇨ Dễ thấy khi E = 1, Q = D, nói cách khác, Output = Input, Latch này như tàng hình, gọi là Transparent Latch

⇨ JK Latch

⇨ Bảng thực trị

J	K	Q	Q’
0	0	0	1
		1	0
0	1	0	1
1	0	1	0
1	1		

⇨ Khi J = K = 1 thì nó sẽ nhảy qua lại liên hồi không nghỉ giữa Q = 0, Q’ = 1 và Q = 1, Q’ = 0 cho đến khi J và K không đồng thời = 1 nữa

⇨ Gated JK Latch

⇨ Khi E = 1, thì = JK Latch, khi E = 0 thì như khi E = 1 và J = K = 0

26. Flip Flop?

Xem cấu trúc các Flip Flop tại “Exercise\he_thong_so\latch_and_flip_flop_and_ic.circ”

⇨ Mỗi Flip Flop đều sẽ có độ trễ lan truyền, nên Input thay đổi thì Output chưa thay đổi ngay

⇨ SR Flip Flop = Gated SR Latch nhưng đầu tiên E sẽ được đi qua 1 Edge Detector

⇨ D Flip Flop = Gated D Latch nhưng đầu tiên E sẽ được đi qua 1 Edge Detector, có thể tạo 1 D Flip Flop từ SR Flip Flop hoặc JK Flip Flop = cách dùng cổng NOT

⇨ Master Slave D Flip Flop = D Flip Flop + SR Flip Flop đảo E

⇨ Bảng thực trị tích cực mức cao

E	E trước	D trước	Q	Q’
0	0		0	1
			1	0
	1	0	0	1
	1	1	1	0
1			0	1
			1	0

⇨ Bảng thực trị tích cực mức thấp

E	E trước	D trước	Q	Q’
0	0		0	1
			1	0
	1	0	1	0
	1	1	0	1
1			0	1
			1	0

⇨ Có tác dụng tạo đỗ trễ khi lưu dữ liệu

⇨ JK Flip Flop = Gated JK Latch nhưng đầu tiên E sẽ được đi qua 1 Edge Detector

⇨ T Flip Flop = JK Flip Flop nhưng J và K hợp thành 1 là T, có tác dụng chuyển đổi qua lại giữa các trạng thái

⇨ Flip Flop với ngõ Clear và Preset

⇨ Bảng thực trị với ngõ Clear và Preset tích cực mức thấp

Clear	Preset	Q	Q’
0	0	1	1
0	1	0	1
1	0	1	0
1	1	0	1
		1	0

27. T Setup Và T Hold?

⇨ Ví dụ T Setup và T Hold của D Flip Flop Clock cạnh lên lần lượt là 1ns và 2ns, thì tín hiệu D cần phải ổn định trong khoảng thời gian từ trước 1ns so với cạnh lên Clock tới 2ns sau cạnh lên Clock, tức là giá trị của nó được giữ không đổi trong khoảng thời gian này, thì Output của Flip Flop mới chắc chắn, nếu trong khoảng thời gian này mà nó chuyển từ 0 lên 1 hay bất ổn định gì đó, thì Output không chắc chắn là 0 hay 1

28. Mạch Đếm (Counter)?

⇨ Mạch đếm bất đồng bộ (Asynchronous Counter)

⇨ Trường hợp 1, đếm lên từ 0 đến $2^N - 1$ rồi ngay lập tức về 0 hoặc ngược lại đếm xuống

⇨ Dùng JK, D hoặc T Flip Flop, Clock cạnh xuống

⇨ Bước 1, lấy N cái xếp thành hàng ngang

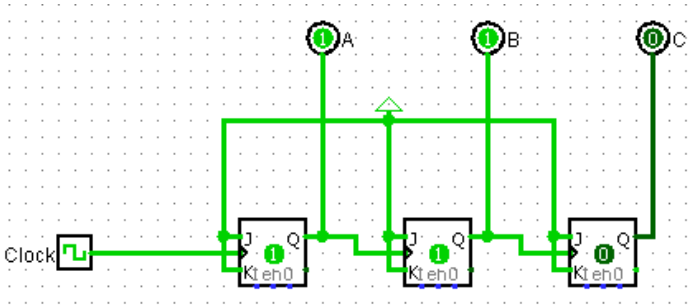
⇨ Bước 2, nối Clock vào E thằng trái cùng

⇨ Bước 3, thiết lập trạng thái cho từng Flip Flop sao cho khi E của nó kích thì Q của nó sẽ Toggle, JK, T thì nối vào mức 1, D thì nối vào Q’ của chính nó

⇨ Bước 4, nối đầu ra thằng bên trái vào E thằng bên phải, nếu đếm lên thì trái dấu, nghĩa là Q nối vào E kích cạnh xuống hoặc Q’ vào E kích cạnh lên, nếu đếm xuống thì cùng dấu

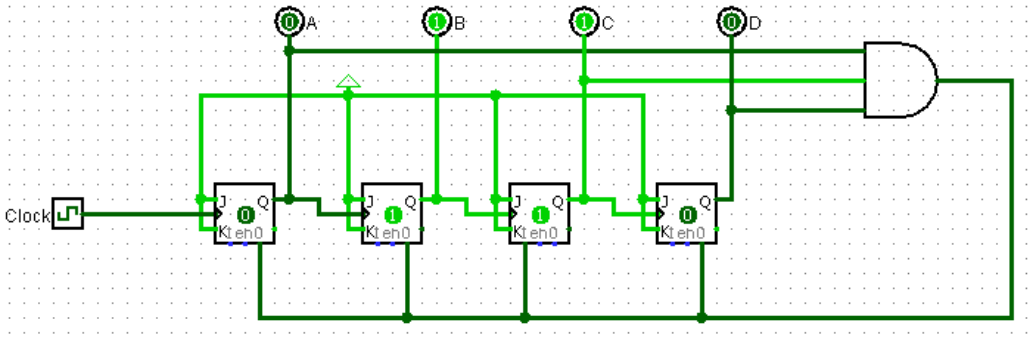
⇨ Bước 5, đọc kết quả, thằng nối trực tiếp với Clock là LSB

⇨ Minh họa

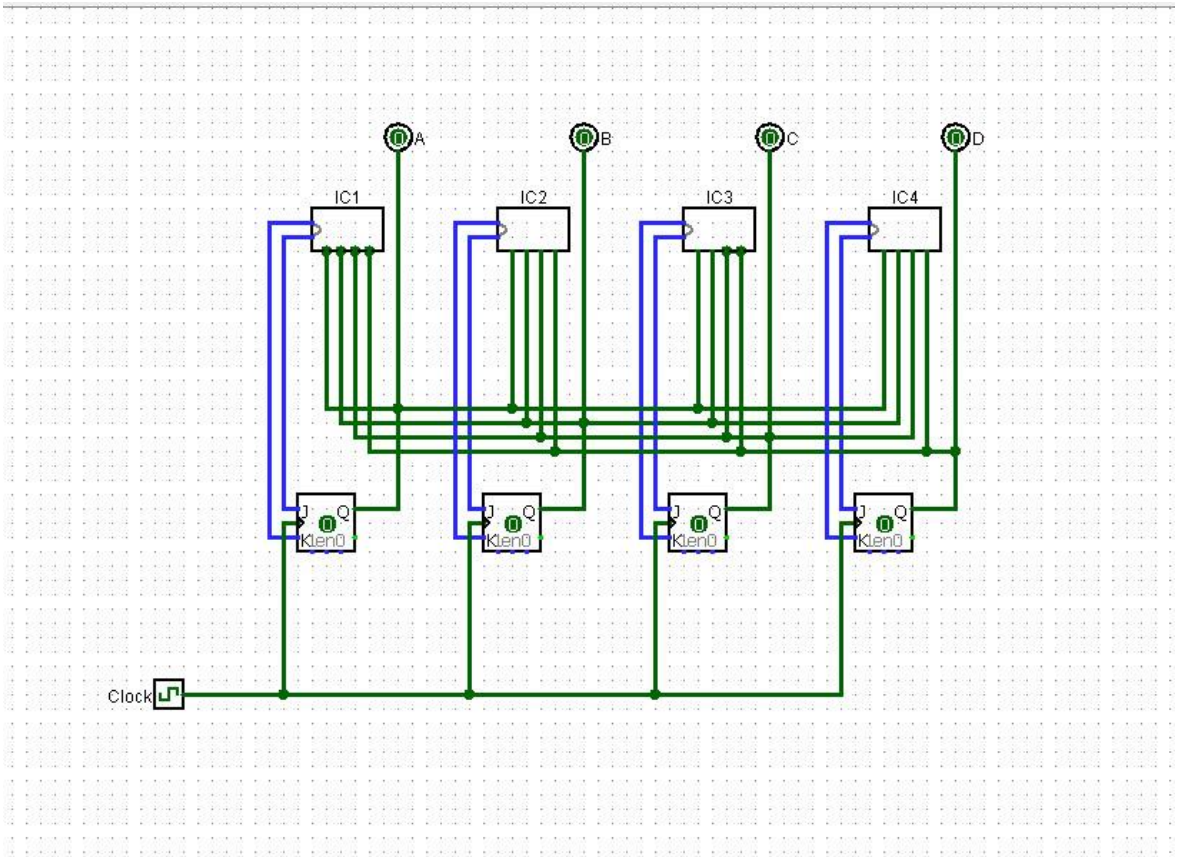


⇨ Cứ mỗi lần Clock cạnh xuống, CBA tăng thêm 1

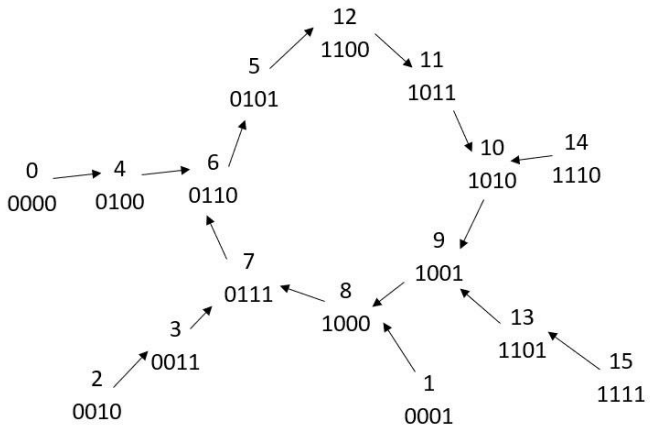
- ⇨ CBA = 000, 001, 010, 011, 100, …, 111, 000, 001, …
- ⇨ Trường hợp 2, đếm lên từ k tới m – 1, m – 1 > k, rồi ngay lập tức về k hoặc ngược lại đếm xuống
- ⇨ Bước 1, xác định số Flip Flop cần sử dụng là N = $\log_2(m)$ làm tròn lên
- ⇨ Bước 2, = bước 1 đến 4 trường hợp 1
- ⇨ Bước 3, nếu đếm lên thì đổi m sang nhị phân, nối các ngõ Q ứng với Bit 1 vào 1 cổng AND hoặc NAND tùy tích cực, nếu đếm xuống thì đổi k – 1 sang nhị phân, nối các ngõ Q ứng với Bit 0 vào 1 cổng OR hoặc NOR tùy tích cực
- ⇨ Bước 4, nếu đếm lên thì đổi k sang nhị phân, nếu đếm xuống thì đổi m – 1 sang nhị phân, nối Output cổng ở bước 3 với Preset hay Clear của từng Flip Flop dựa vào dạng nhị phân này, nếu Bit 0 thì Clear, Bit 1 thì Preset
- ⇨ Minh họa mạch đếm từ 4 lên 12 về 4



- ⇨ Cứ mỗi lần Clock cạnh xuống, DCBA tăng 1
- ⇨ DCBA = 0100, 0101, 0110, 0111, …, 1100, 0100, 0101, …
- ⇨ Do khi DCBA ở 1100, 1 lát sau lên 1101 là trở về 0100 ngay, ta có A từ 0 lên 1 rồi ngay lập tức xuống 0, do đó tại A và Output cổng AND xuất hiện xung gai
- ⇨ Khi tạo mạch đếm theo kiểu bất đồng bộ, giả sử độ trễ lan truyền mỗi Flip Flop = t giây, thì Flip Flop thứ n từ trái sang sẽ bị chậm nt giây so với cạnh xuống Clock
- ⇨ Chu kì tối thiểu của Clock để mạch đếm hoạt động tạm gọi là bình thường và không bị quá lệch so với Clock = Nt, N là số Flip Flop
- ⇨ Nếu trạng thái ban đầu của bộ đếm vượt ngoài khoảng đếm, ví dụ đếm từ 4 tới 12 mà giá trị ban đầu = 13, thì nó sẽ tiếp tục đếm 13, 14, 15, 0, 1, 2, 3, 4 rồi bị nhốt vào vòng lặp 4 tới 12
- ⇨ Mạch đếm đồng bộ (Synchronous Counter)
- ⇨ Giả sử ta muốn đếm từ 12 xuống 5 rồi quay ngược lại 12
- ⇨ Dùng JK Flop
- ⇨ Bước 1, xác định số Bit tối thiểu để biểu diễn các số từ 12 đến 5, ta có 12 = 1100, vậy cần 4 Bit hay 4 Flip Flop
- ⇨ Bước 2, xếp 4 Flip Flop thành hàng ngang, nối Q của từng thằng ra Output A, B, C, D, trong đó A là MSB và D là LSB, đồng thời nối chung E của chúng với Clock, và cho 1 mỗi Flip Flop 1 IC có 4 chân Input và 2 chân Output, 4 Input được nối với ABCD, 2 Output nối vào JK
- ⇨ Minh họa



- ⇨ Bước 3, lập sơ đồ chuyển đổi trạng thái, ví dụ ABCD đang là 1100 thì trạng thái tiếp theo khi có xung Clock sẽ là 1011, dùng mũi tên để chỉ sự chuyển đổi, nếu các giá trị nằm ngoài vùng đếm thì nối nó vào vùng đếm luôn, nối kiểu nào cũng được
- ⇨ Minh họa



- ⇨ Bước 4, dựa vào sơ đồ này để xác định cấu trúc mạch của các IC ứng với từng JK Flip Flop
- ⇨ Ví dụ xét IC1, giả sử ban đầu ABCD = 0000, ta muốn lần xung Clock kế tiếp nó thành 0100 như trong sơ đồ, mà IC1 chịu trách nhiệm tạo JK cho JK Flip Flop đầu tiên, từ đó xuất ra A, mà A ở đây = 0 chuyển sang 0, điều này chỉ xảy ra khi

J = 0 và K = bất kì, như vậy ứng với ABCD = 0000, 2 chân Output JK của IC1 phải = 0x, tương tự với ABCD = 0001, 0010, …, và tương tự với các IC khác

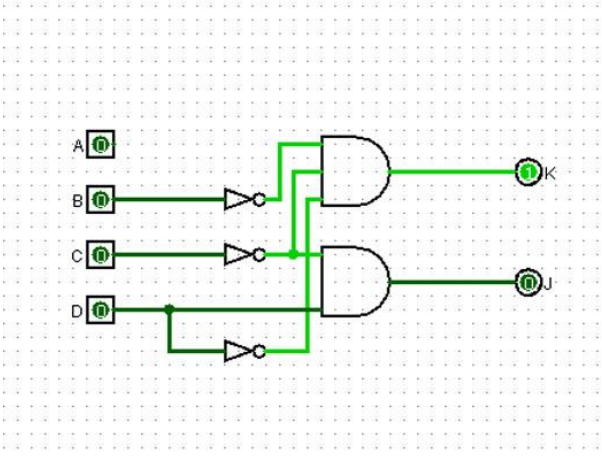
⇨ Từ đó, ta lập được bảng thực trị của tất cả IC

Đếm	A	B	C	D	J_{IC1}	K_{IC1}	J_{IC2}	K_{IC2}	J_{IC3}	K_{IC3}	J_{IC4}	K_{IC4}
0	0	0	0	0	0	x	1	X	0	x	0	x
1	0	0	0	1	1	x	0	X	0	x	x	1
2	0	0	1	0	0	x	0	X	x	0	1	x
3	0	0	1	1	0	x	1	X	x	0	x	0
4	0	1	0	0	0	x	x	0	1	x	0	x
5	0	1	0	1	1	x	x	0	0	x	x	1
6	0	1	1	0	0	x	x	0	x	1	1	x
7	0	1	1	1	0	x	x	0	x	0	x	1
8	1	0	0	0	x	1	1	X	1	x	1	x
9	1	0	0	1	x	0	0	X	0	x	x	1
10	1	0	1	0	x	0	0	X	x	1	1	x
11	1	0	1	1	x	0	0	X	x	0	x	1
12	1	1	0	0	x	0	x	1	1	x	1	x
13	1	1	0	1	x	0	x	1	0	x	x	0
14	1	1	1	0	x	0	x	1	x	1	0	x
15	1	1	1	1	x	0	x	0	x	0	x	0

⇨ Tiến hành dùng biểu đồ Karnaugh để giải, thu được các kết quả sau

J_{IC1}	C'D	C'D
K_{IC1}	B'C'D'	B'C'D'
J_{IC2}	C'D' + A'CD	C'D' + A'CD
K_{IC2}	AC' + AD'	A(C' + D')
J_{IC3}	BD' + AD'	D'(A + B)
K_{IC3}	BD' + AD'	D'(A + B)
J_{IC4}	A'C + AB' + AC'	AB' + (A EXOR C)
K_{IC4}	A'C' + A'B + AB'	A'C' + (A EXOR B)

⇨ Như vậy là ta đã dễ dàng có thể lập được sơ đồ mạch, ví dụ bên trong IC1 sẽ có cấu trúc như sau



⇨ Các IC khác tương tự, như vậy là ta đã tạo thành công 1 mạch đếm từ 12 xuống 5

⇨ Lưu ý do đã có sẵn cổng Q', nên để có A', B', C', D' thì thay vì tốn thêm 1 cổng NOT ta có thể dùng Q' của mỗi Flip Flop luôn

⇨ Ưu điểm của mạch đếm đồng bộ là độ trễ lan truyền không cộng dồn theo số lượng Flip Flop, độ trễ lan truyền tổng cộng = độ trễ lan truyền của 1 Flip Flop + độ trễ lan truyền của mạch tổ hợp có độ trễ lớn nhất, do độ trễ lan truyền của

mạch là khoảng thời gian từ lúc Clock kích cho đến Output của tất cả các cổng ổn định, bao gồm cả cổng trong mạch tổ hợp

⇨ Để xác định 1 mạch đếm đồng bộ đếm từ đâu đến đâu, thực hiện các bước

⇨ Bước 1, lập công thức theo A, B, C, … của các chân J, K

⇨ Bước 2, vẽ bảng thực trị, với mỗi tổ hợp A, B, C, … sẽ cho 1 giá trị J, K khác nhau

⇨ Bước 3, dựa vào giá trị của J, K để xác định trạng thái tiếp theo của mạch đếm

⇨ Một số mạch đếm đồng bộ quen thuộc dùng JK Flip Flop, A_1 là LSB, A_N là MSB

⇨ Mạch đếm từ 0 tới $2^N - 1$ rồi ngay lập tức về 0

J_{IC1}	1
K_{IC1}	1
J_{IC2}	A_1
K_{IC2}	A_1
J_{IC3}	A_1A_2
K_{IC3}	A_1A_2
J_{IC4}	$A_1A_2A_3$
K_{IC4}	$A_1A_2A_3$
…	…
J_{ICN}	$A_1A_2A_3…A_{N-1}$
K_{ICN}	$A_1A_2A_3…A_{N-1}$

⇨ Ở đây nhận thấy, ta có thể xây dựng mạch đếm lên đồng bộ từ a tới b bất kì sử dụng cấu trúc của mạch đếm lên này, bằng cách tận dụng thêm chân Clear và Preset

⇨ Ví dụ trong lúc nó đang đếm lên từ 000 tới 111, ta muốn khi đạt 110 thì Reset về 001, cách tạo mạch cho Clear và Preset như ở mạch đếm bất đồng bộ

⇨ Mạch đếm từ $2^N - 1$ xuống 0 rồi ngay lập tức về $2^N - 1$

J_{IC1}	1
K_{IC1}	1
J_{IC2}	A_1'
K_{IC2}	A_1'
J_{IC3}	$A_1'A_2'$
K_{IC3}	$A_1'A_2'$
J_{IC4}	$A_1'A_2'A_3'$
K_{IC4}	$A_1'A_2'A_3'$
…	…
J_{ICN}	$A_1'A_2'A_3'…A_{N-1}'$
K_{ICN}	$A_1'A_2'A_3'…A_{N-1}'$

⇨ Ta cũng có thể sử dụng cấu trúc này để đếm xuống từ a xuống b bất kì, ý tưởng giống ở trên

⇨ Giả sử mạch đếm 0 đến k – 1, dễ thấy MSB của mạch đếm có tần số = tần số của Clock / k, nên mạch đếm còn được dùng làm mạch giảm tần số

29. Cách Nhanh Nhất Để Phân Tích Mạch Đếm Đồng Bộ Hoặc Tạo Ra Nó?

⇨ Để phân tích xem 1 mạch đếm đồng bộ đếm từ đâu đến đâu, ở đây ta sẽ xét bộ đếm 4 Bit dùng JK Flip Flop, D là MSB, A là LSB

⇨ Cách 1

⇨ Bước 1, lập công thức các chân J, K, viết kiểu gì cũng được

⇨ Bước 2, ứng với mỗi công thức, ghi biểu diễn nhị phân tương ứng từ MSB tới LSB, ví dụ nếu công thức là DCA’ thì ghi 11x0, nếu công thức là BA + C thì ghi xx00 + x1xx

⇨ Bước 3, vẽ 1 biểu đồ Karnaugh 4 x 4 trống, chia mỗi ô trong biểu đồ thành 2 hàng 4 cột = 8 ô nhỏ để điền giá trị của 8 chân J, K, thứ tự điền như sau

J_D	K_D	J_C	K_C
J_B	K_B	J_A	K_A

⇨ Bước 4, lần lượt xét các chân, với mỗi chân, điền Full biểu đồ Karnaugh ứng với chân của nó, chỉ điền 1, nếu 0 thì để trống

⇨ Bước 5, lập giản đồ chuyển đổi trạng thái, bắt đầu từ 0000, nhìn vào ô ứng với 0000, dựa vào giá trị 8 chân J, K để xác định trạng thái tiếp theo, sau đó đánh dấu ô 0000, nhảy tới ô có giá trị = trạng thái tiếp theo và làm tương tự

⇨ Cách 2, dùng CASIO

⇨ A OR B = M – |A + B – M|, M = 1.5

⇨ A OR B OR C = M – |A – |B + C – M||

⇨ A OR B OR C OR D = M – |A – |B – |C + D – M|||

⇨ AND = AB

⇨ EX – OR = |A – B|

⇨ NOT = 1 – A

⇨ Để lập công thức cho các chân khi có giản đồ chuyển đổi trạng thái, ở đây ta sẽ xét giản đồ có giá trị từ 0 đến 15, và ta dùng JK Flip Flop, D là MSB, A là LSB

⇨ Bước 1, vẽ biểu đồ Karnaugh 4 x 4 trống

⇨ Bước 2, học thuộc 000x, 011x, 10x1, 11x0

⇨ Bước 3, chọn 1 Node trong giản đồ chuyển đổi trạng thái, gọi là A, xem giá trị nó trỏ tới, gọi là B, ví dụ Node này = 1010 trỏ tới 1100

⇨ Bước 4, trong biểu đồ Karnaugh, tại ô ứng với A, chia làm 2 hàng 4 cột = 8 ô để điền giá trị của 8 chân J, K, thứ tự điền như sau

J_D	K_D	J_C	K_C
J_B	K_B	J_A	K_A

⇨ Bước 5, dựa vào trạng thái tiếp theo, điền Full giá trị vào ô này, chỉ điền 1 và x, 0 để trống

⇨ Bước 6, quay lại bước 2, làm cho tới khi điền Full ô

⇨ Bước 7, từ biểu đồ Karnaugh 8 chân, lần lượt trích xuất biểu đồ Karnaugh của từng chân và viết biểu thức rút gọn

30. Mạch Đếm Vòng (Ring Counter)?

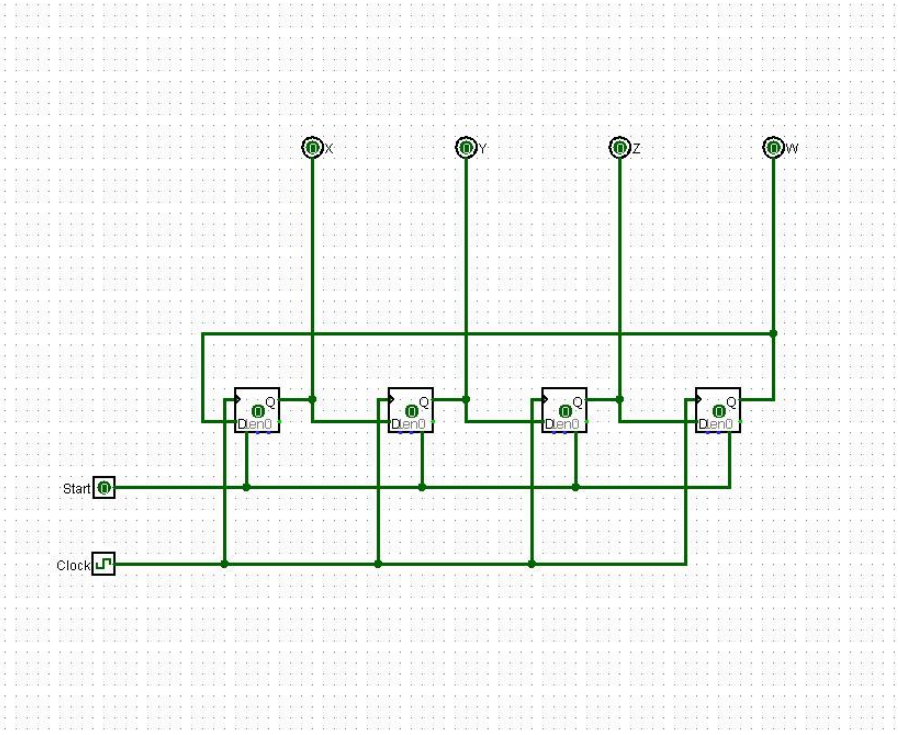
⇨ Là mạch đếm kiểu dịch Bit sang phải với mỗi xung Clock, Bit cuối sẽ dời lên Bit đầu, ví dụ giá trị hiện tại là 0110 thì tiếp theo sẽ là 0011, và tiếp nữa là 1001, rồi 1100, và quay lại 0110, …, hay nếu giá trị hiện tại là 0111 thì giá trị tiếp theo là 1011, … và khi Start đạt mức điện thế tích cực, ví dụ 1 nếu là tích cực mức cao, là mạch bị Reset về giá trị tùy theo mình nối Start vào Preset hay Clear của từng chân, ví dụ nối Start vào Preset của Bit thứ 2 và 3, còn lại nối Clear, thì khi Start tích cực, giá trị của mạch sẽ là 0110 và tiếp túc đếm từ giá trị này khi Start hết tích cực

⇨ Ví dụ để tạo mạch đếm vòng dùng D Flip Flop, đếm từ 1000, 0100, …

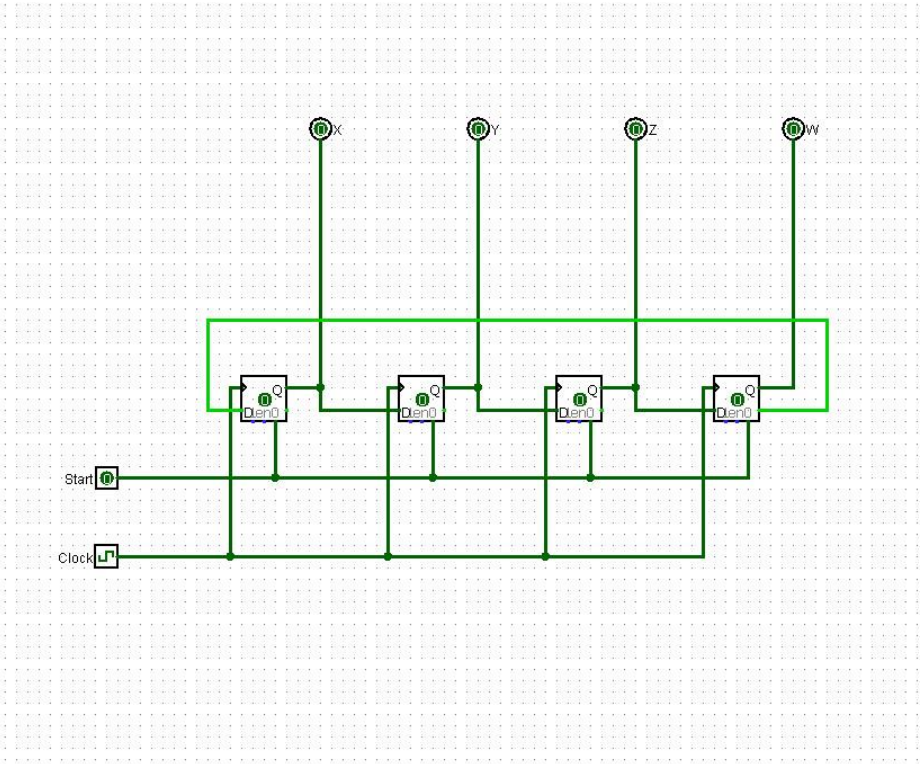
⇨ Bước 1, xác định số Bit, số này = số D Flip Flop, ở ví dụ trên là 4

⇨ Bước 2, xếp 4 D Flip Flop thành hàng ngang, nối Q của từng thằng ra Output X, Y, Z, W, đồng thời nối chung E của chúng với Clock, và nối Q của thằng trước với D của thằng sau, cuối cùng nối Preset của X và Clear của Y, Z, W với Start

⇨ Minh họa



- ⇨ Mạch đếm Johnson = mạch đếm vòng nhưng giá trị đảo của Bit cuối sẽ truyền cho Bit đầu, ví dụ 0110, 1011, 0101, 0010, 1001, 0100, 1010, 1101, 0110, …
- ⇨ Cấu trúc tương tự mạch đếm vòng, Q đằng trước nối D đằng sau, nhưng Q’ của thằng cuối thay vì Q sẽ nối với D của thằng đầu
- ⇨ Minh họa mạch đếm 0000, 1000, 1100, 1110, 1111, 0111, 0011, 0001, 0000, …



31. MUX (Multiplexer)?

⇨ Là 1 IC, có tác dụng chọn 1 trong các giá trị đầu vào để trả về, ví dụ có 4 giá trị đầu vào là 0, 1, 1, 0, ta muốn chọn giá trị có Index = 2, như vậy kết quả trả về phải = 1

⇨ Như vậy MUX sẽ có k cổng Input, $\log_2 k$ cổng Index, lưu ý làm tròn lên, và 1 cổng Output

⇨ Áp dụng công thức dạng chuẩn hội hoặc chuẩn tuyển để tìm sơ đồ mạch điện

⇨ Ví dụ, MUX có 4 Input I_0, I_1, I_2, I_3, do đó sẽ có 2 cổng Index S_0, S_1, gọi cổng Output là X, ta có công thức

$$X = S_1'S_0'I_0 + S_1'S_0I_1 + S_1S_0'I_2 + S_1S_0I_3$$

⇨ Dễ thấy khi $S_0S_1 = 00$, thì $X = I_0$, $S_0S_1 = 01$, thì $X = I_1$, $S_0S_1 = 10$, thì $X = I_2$, $S_0S_1 = 11$, thì $X = I_3$, …

32. Dùng MUX Để Lập Mạch Cho Biểu Thức Boolean Bất Kì?

⇨ Giả sử ta muốn dùng 4 to 1 MUX để lập mạch ứng với hàm F(A, B, C, D)

⇨ Bước 1, vẽ biểu đồ Karnaugh của F

1	1		
		1	
	1	1	
			1

⇨ Bước 2, xếp các biến bắt đầu từ MSB vào các chân Select, $A = S_1$, $B = S_2$

⇨ Bước 3, lặp từ trên xuống, bắt đầu từ I_0, ứng với dòng 00 trong biểu đồ Karnaugh

00	01	11	10
1	1		

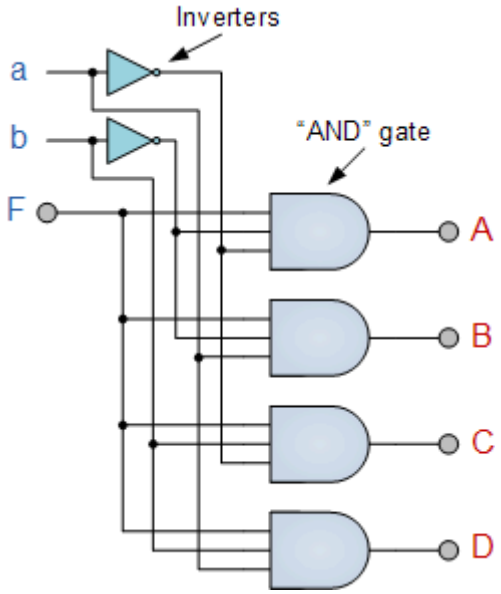
⇨ Dòng này ứng với biểu thức C', do đó nối C' vào I_0

⇨ Tương tự với I_1, …

33. DEMUX (Demultiplexer)?

⇨ Là 1 IC, có tác dụng truyền dữ liệu đầu vào cho 1 trong các dây Output, nghịch đảo của MUX

⇨ Ví dụ



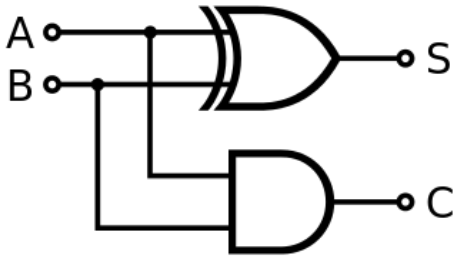
⇨ Bảng thực trị mạch trên

a	b	A	B	C	D
0	0	F	0	0	0
0	1	0	0	F	0
1	0	0	F	0	0

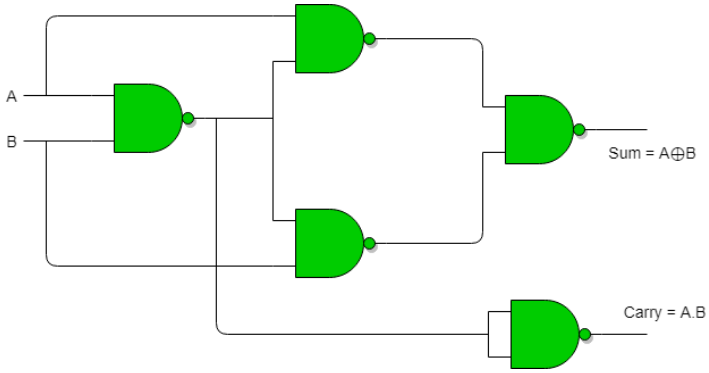
1	1	0	0	0	F

34. Mạch Cộng (Adder)?

⇨ Mạch cộng nửa (Half Adder) cộng 1 Bit với 1 Bit ra tổng và phần nhớ



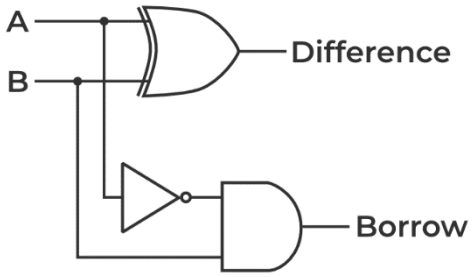
⇨ Biểu diễn Full NAND



⇨ Bảng thực trị

A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

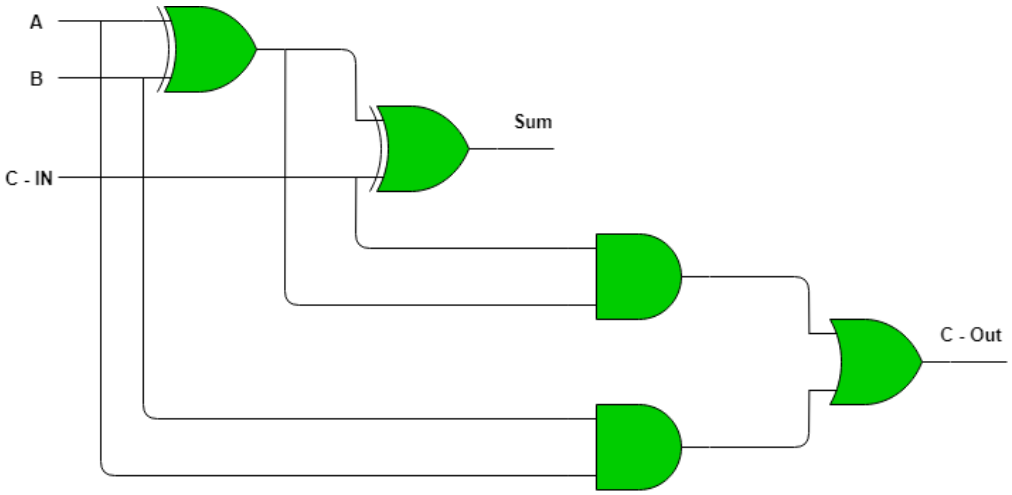
⇨ Mạch trừ nửa (Half Subtractor) tính Bit A – Bit B ra hiệu và phần mượn



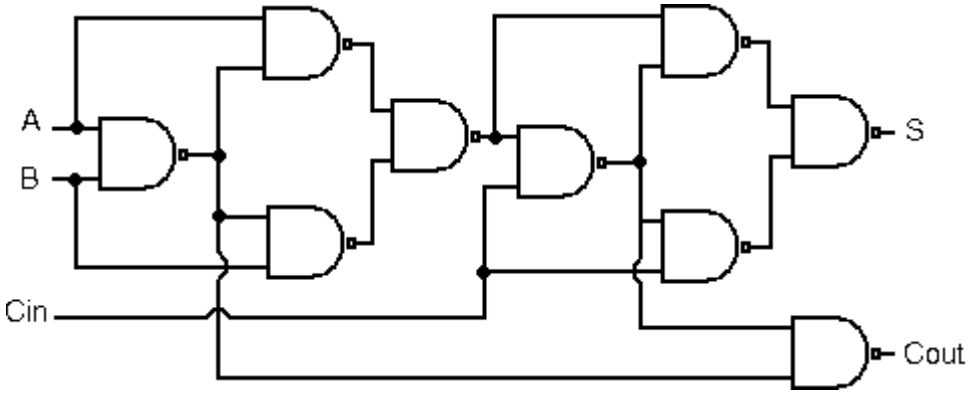
⇨ Bảng thực trị

A	B	S	C
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

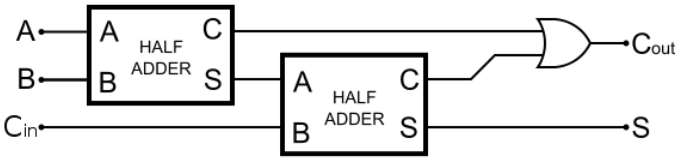
⇨ Mạch cộng Full (Full Adder) = mạch cộng nửa nhưng thêm phần nhớ trước



⇨ Biểu diễn Full NAND



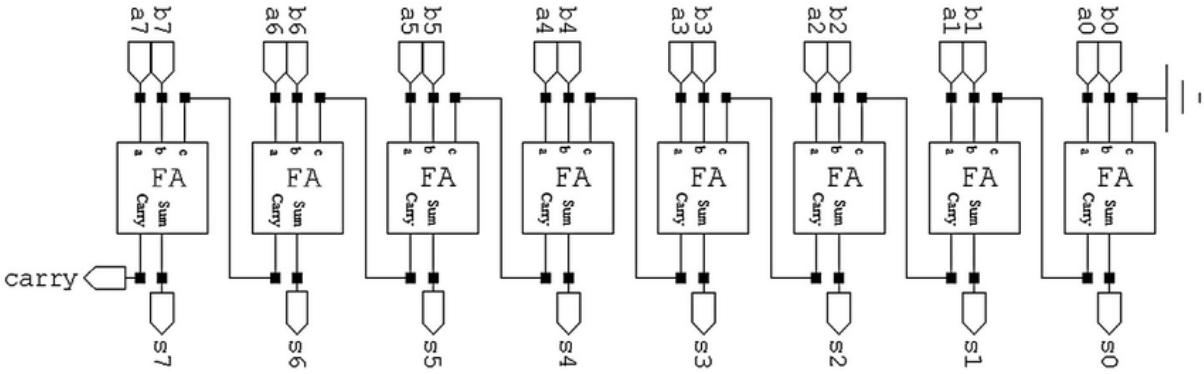
⇨ Dùng 2 mạch cộng nửa mắc nối tiếp



⇨ Bảng thực trị

A	B	C_{IN}	S	C_{OUT}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

⇨ Mạch cộng 8 Bit với 8 Bit (8 Bit Ripple Carry Adder) ví dụ
10011110 + 11001101 = 101101011



⇨ Ở đây dùng 8 mạch cộng Full mắc nối tiếp, $a_7a_6…a_0 + b_7b_6…b_0 = \text{carry } s_7s_6…s_0$

⇨ Mạch cộng tính dư trước (Carry Look Ahead Adder)

⇨ Mạch này khắc phục nhược điểm của mạch cộng trên khi càng nhiều mạch cộng Full được mắc nối tiếp với nhau, độ trễ lan truyền càng lớn

⇨ Giả sử số dư khởi đầu trước khi cộng là C, gọi C_0 là số dư tại mạch cộng Full thứ nhất, gọi $G_n = A_nB_n$, $P_n = A_n$ EX – OR B_n, theo sơ đồ mạch cộng Full, ta có

$$C_0 = G_0 + P_0 \cdot C$$

⇨ Từ đây ta có thể tính được C_1 trực tiếp từ C

$$C_1 = G_1 + P_1 \cdot C_0 = G_1 + P_1 \cdot (G_0 + P_0 \cdot C) = G_1 + P_1 \cdot G_0 + P_1 \cdot P_0 \cdot C$$

⇨ Và cứ như thế, ta tính được C_n trực tiếp từ C

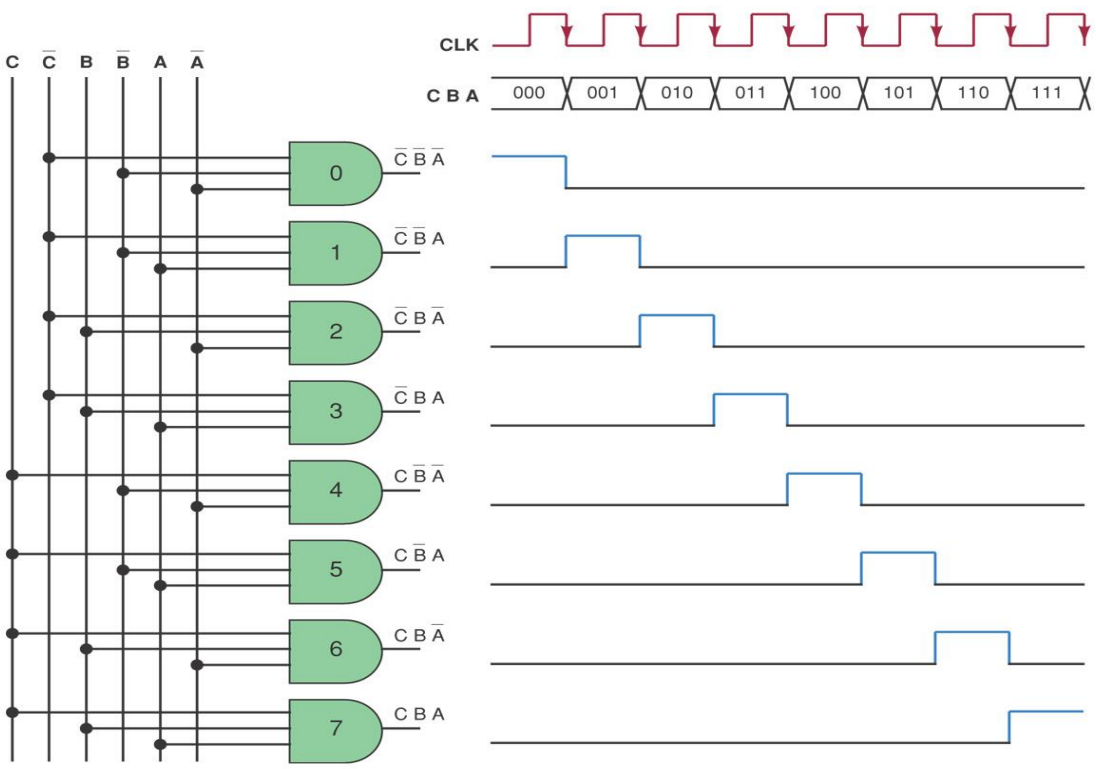
⇨ Để ý thấy, với mỗi C_n, ta sẽ phải lập 1 mạch tổ hợp với Input là các giá trị P, G đã tính trước, cùng với giá trị C khởi đầu, vì biểu thức của C_n có dạng SOP, ta sẽ dùng 1 loạt các cổng AND nhiều ngõ nhập, sau đó cho tất cả qua 1 cổng OR, như vậy chỉ có 2 khoảng bị cộng dồn độ trễ thời gian, bù lại dùng nhiều cổng vãi cả lồn

⇨ Sau khi có tất cả C_n thì dựa vào bảng thực trị mạch cộng Full để tính S_n

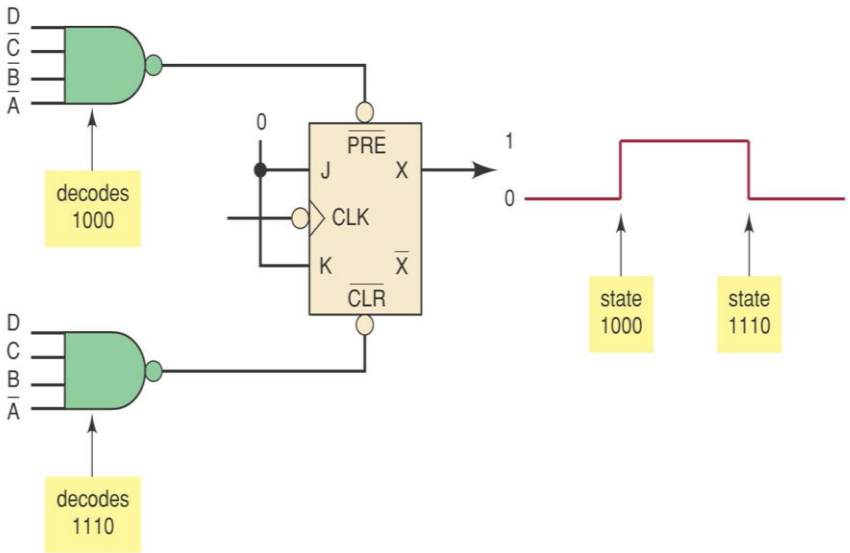
35. Decoder?

⇨ Đầu vào là tín hiệu nhị phân, ví dụ 101, đầu ra là tín hiệu số nguyên, ví dụ 5, 5 tức là đèn số 5 sẽ sáng

⇨ Sử dụng SOP để xây dựng mạch, ví dụ



⇨ Giả sử ta có 1 mạch đếm, trả về từ 0000 đến 1111, ta muốn trong khoảng 1000 đến 1110 thì đèn sẽ sáng, còn lại đèn tắt, sử dụng mạch sau, D là MSB, A là LSB



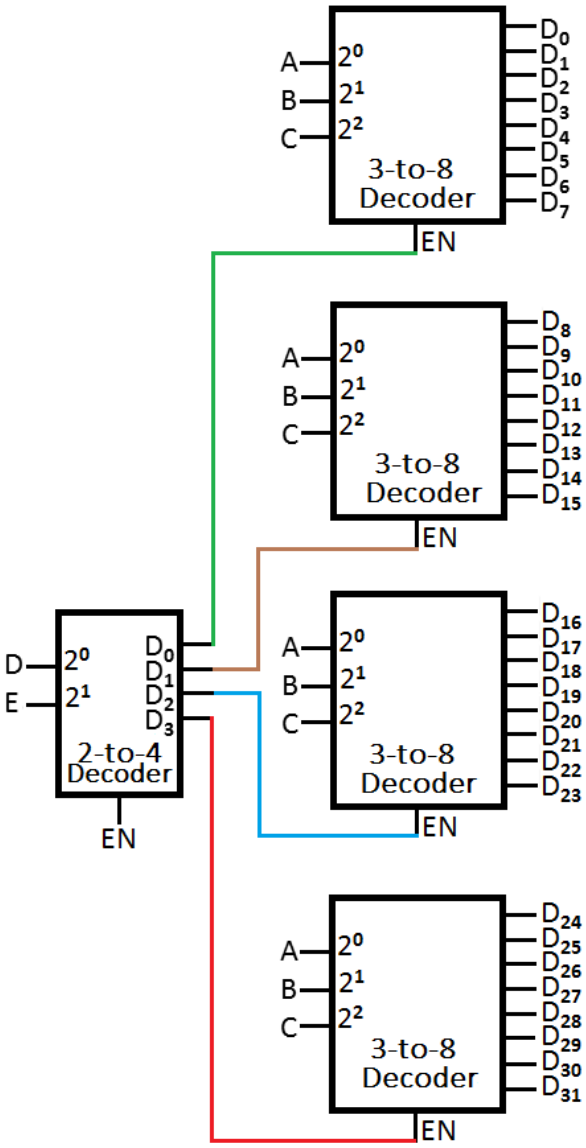
⇨ Giả sử ta muốn tạo 1 mạch k to 2^k Decoder từ n to 2^n Decoder

⇨ Bước 1, lấy 2^{k-n} cái n to 2^n Decoder xếp thành hàng dọc, Output từ trên xuống sẽ là 0, 1, 2, …

⇨ Bước 2, lấy 1 cái k – n to 2^{k-n} Decoder, nối Output của nó vào cổng Enable của từng cái n to 2^n, Enable mức 0 thì tất cả Output của Decoder đó = 0

⇨ Bước 3, tất cả n to 2^n Decoder đều sử dụng chung Input, vậy tổng cộng có k Input và 2^k Output

⇨ Ví dụ 5 to 32 Decoder từ 3 to 8 Decoder



36. Dùng Decoder Để Tạo Mạch Boolean Bất Kì?

⇨ Giả sử ta muốn ta muốn tạo mạch ứng với hàm F(A, B, C, D) dùng Decoder tích cực mức cao

⇨ Bước 1, chọn Decoder với số Input = số biến = 4, mắc 4 biến vào 4 Input

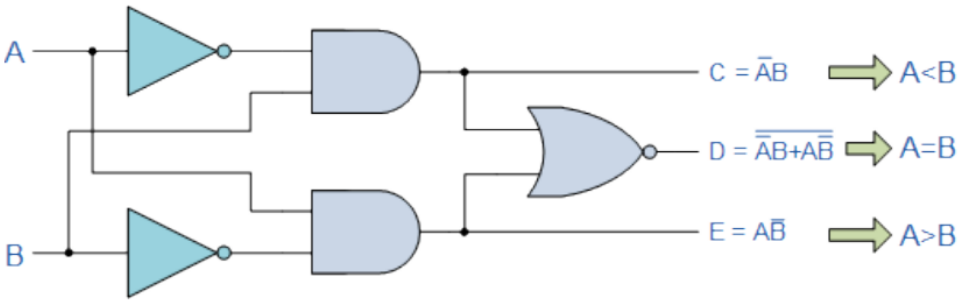
⇨ Bước 2, xác định các tổ hợp A, B, C, D hàm trả về 1, nối các Output ứng với các tổ hợp này vào 1 cổng OR duy nhất, Output cổng OR là Output cuối cùng

37. Encoder?

⇨ Là mạch có 2^k Input và k Output, vì dụ Input là 00000010 với chỉ có Bit 6 tích cực thì Output = 110 = 6

38. Mạch So Sánh (Comparator)?

⇨ Để so sánh 2 Bit xem thằng nào lớn, bé hoặc bằng



39. Shift Register?

⇨ Khi viết chuỗi Bit trong Shift Register, ta viết Bit trái cùng là MSB còn phải cùng là LSB

⇨ Khi nói dịch trái thì ta sẽ dịch trái chuỗi trên, tương tự dịch phải

⇨ Đầu ra là Bit bị đẩy ra, đầu vào là Bit bị nhồi vào

Logic:

1. Mệnh Đề (Proposition, Statement)?

⇨ Là 1 phát biểu trả về giá trị đúng hoặc sai tại 1 thời điểm xác định, không được vừa đúng vừa sai

⇨ Ví dụ “n > 100” không phải là mệnh đề vì nó có thể đúng hoặc sai tùy theo n, “hôm nay là thứ 3” không phải mệnh đề vì thời điểm phát biểu được tạo ra không được xác định, “$n^2 + 1 > 0$” là mệnh đề vì với mọi n, bất đẳng thức này đều đúng

2. Hội (Conjunction)?

$p \wedge q$

⇨ Là phép AND của 2 mệnh đề

⇨ Mệnh đề này tương đương với các mệnh đề sau

p và q p nhưng q

⇨ Kí hiệu phép NAND

$p \mid q$

3. Tuyển (Disjunction)?

$p \vee q$

⇨ Là phép OR của 2 mệnh đề, tương đương các mệnh đề sau

“Hoặc là p, hoặc là q”

⇨ Kí hiệu phép NOR

$p \downarrow q$

4. Tuyển Loại (Exclusive Or)?

$p \oplus q$

⇨ Là phép EX – OR của 2 mệnh đề

5. Điều Kiện Cần Và Đủ?

⇨ Ví dụ xét sự kiện A là n chia hết cho 10

⇨ n chia hết cho 20 chính là điều kiện đủ, vì chỉ cần nó thỏa mãn là chắc chắn A xảy ra

⇨ n chia hết cho 5 chính là điều kiện cần, vì để n chia hết cho 10 thì nó phải chia hết cho 5 trước đã

6. Kéo Theo (Implication)?

$p \to q = \bar{p} + q$

⇨ Mệnh đề này tương đương với các mệnh đề sau

Nếu p, thì q
Nếu p, q
Khi p, q
p nên q
p kéo theo q
p suy ra q
p là điều kiện đủ để q
q là điều kiện cần để p
q bất cứ khi nào p
Nó là điều kiện cần để q bất cứ khi nào p
q nếu p
p chỉ khi q
Chỉ p khi q
p chỉ nếu q
q trừ khi p’
q, ngoại trừ p’
p là q

⇨ Ví dụ

⇨ Xét mệnh đề “Nếu tôi địt, thì bạn rên”, các mệnh đề tương đương là

“Nếu tôi địt, bạn rên”
“Khi tôi địt, bạn rên”
“Tôi địt nên bạn rên”
“Tôi địt kéo theo bạn rên”
“Tôi địt suy ra bạn rên”
“Tôi địt là điều kiện đủ để bạn rên”
“Bạn rên là điều kiện cần để tôi địt”
“Bạn rên bất cứ khi nào tôi địt”
“Nó là điều kiện cần để bạn rên bất cứ khi nào tôi địt”
“Bạn rên nếu tôi địt”
“Tôi địt chỉ khi bạn rên”
“Tôi chỉ địt khi bạn rên”
“Tôi địt chỉ nếu bạn rên”
“Bạn rên trừ khi tôi không địt”
“Bạn rên, ngoại trừ tôi không địt”
“Tôi địt là bạn rên”

⇨ Xét mệnh đề

$(p \to q) \wedge (\overline{q \to p})$

⇨ Mệnh đề này tương đương

“q là điều kiện cần nhưng không đủ để p”

7. Đảo (Converse) Và Phản Đảo (Contrapositive)?

⇨ Xét mệnh đề kéo theo sau

$p \to q$

⇨ Mệnh đề đảo của mệnh đề này là

$q \to p$

⇨ Mệnh đề phản đảo của mệnh đề này là

$\bar{q} \to \bar{p}$

⇨ Mệnh đề phản đảo bản chất tương đương mệnh đề kéo theo ban đầu

8. Tương Đương (Biconditional)?

$p \leftrightarrow q = \overline{p \oplus q}$

⇨ Mệnh đề này tương đương với các mệnh đề sau

p khi và chỉ khi q
p là điều kiện cần và đủ để q
nếu p thì q, và ngược lại

⇨ Ví dụ

⇨ Xét mệnh đề "Nếu tôi địt, thì bạn rên", các mệnh đề tương đương là

"Nếu tôi địt, bạn rên"
"Tôi địt nên bạn rên"
"Tôi địt là điều kiện đủ để bạn rên"
"Bạn rên nếu tôi địt"
"Tôi địt chỉ khi bạn rên"
"Bạn rên trừ khi tôi không địt"

9. Độ Ưu Tiên Các Toán Tử Logic?

Độ ưu tiên	Toán tử
1	Trong ngoặc tròn
2	Phủ định
3	Hội, tuyển, tuyển loại
4	Kéo theo
5	Tương đương

⇨ Cùng chung độ ưu tiên thì thực hiện từ trái sang

10. Hằng Đúng (Tautology) Và Hằng Sai (Contradiction)?

⇨ Hằng đúng là 1 biểu thức của các biến Logic, sao cho nó luôn trả về giá trị đúng với mọi tổ hợp giá trị của các toán hạng

⇨ Ví dụ các hằng đúng

$p \vee \bar{p}$
$p \to p$
$p \wedge q \to p \vee q$

⇨ Hằng sai tương tự, luôn trả về giá trị sai

⇨ Ví dụ các hằng sai

$p \wedge \bar{p}$

⇨ Nếu không phải hằng đúng và hằng sai thi gọi là Contingency

11. 2 Mệnh Đề Bằng Nhau?

⇨ Là khi mệnh đề tương đương của chúng là 1 hằng đúng, khi này viết

$p \equiv q$

⇨ Ví dụ các mệnh đề bằng nhau

$(p \to q) \wedge (p \to r) \equiv p \to (q \wedge r)$
$(p \to q) \vee (p \to r) \equiv p \to (q \vee r)$
$(p \to r) \wedge (q \to r) \equiv (p \vee q) \to r$

$(p \to r) \vee (q \to r) \equiv p \to (q \to r) \equiv (p \wedge q) \to r$
$\overline{p \to q} \equiv p \wedge \bar{q}$
$p \leftrightarrow q \equiv (p \to q) \wedge (q \to p)$

12. Vị Từ (Predicate)?

⇨ Là 1 hàm mà khi Pass Full đối số cho nó sẽ trả về 1 mệnh đề, ví dụ P(x, y) = “x > 3 + y” là 1 vị từ, P(1, 2) = “1 > 3 + 2” là 1 mệnh đề

⇨ Xét mệnh đề

$\forall x P(x)$

⇨ Mệnh đề này tương đương

“P(x) đúng với mọi x”

⇨ 1 giá trị của x được gọi là phản ví dụ (Counterexample) của mệnh đề trên khi P(x) sai, ví dụ P(x) = “x > 1”, thì 0 là phản ví dụ của mệnh đề “x > 1 đúng với mọi x”

⇨ Xét mệnh đề

$\exists x P(x)$

⇨ Mệnh đề này tương đương

“Tồn tại 1 giá trị của x để P(x) đúng”

⇨ Khi thiết lập 1 mệnh đề thuộc 2 dạng trên cần chỉ rõ vị từ, miền giá trị của biến, ví dụ thiết lập mệnh đề “Một số thằng trong làng đã địt vợ tôi”

P(x) = x đã địt vợ tôi
Miền giá trị của x là toàn bộ thằng trong làng
$\exists x P(x)$

⇨ Khi này với mọi x tương đương việc ta lặp qua tất cả phần tử trong miền giá trị của x, chỉ cần 1 phần tử sai thì cả mệnh đề sai, chỉ một x cũng vậy, nhưng chỉ cần 1 phần tử đúng thì cả mệnh đề đúng

⇨ Ta có các công thức

$\overline{\forall x P(x)} \equiv \exists x \overline{P(x)}$
$\overline{\exists x P(x)} \equiv \forall x \overline{P(x)}$
$\forall x\big(P(x) \wedge Q(x)\big) \equiv \forall x P(x) \wedge \forall x Q(x)$
$\exists x\big(P(x) \vee Q(x)\big) \equiv \exists x P(x) \vee \exists x Q(x)$

⇨ Nếu trước vị từ có dấu mọi hoặc tồn tại một, mà đi kèm với dấu đó là biến không có trong vị từ, thì loại bỏ nó

⇨ Ví dụ

$\exists y \forall x \forall z P(x) \equiv \forall x P(x)$

⇨ Các vị từ sau là hằng đúng

$\forall x P(x) \vee \exists x \overline{P(x)}$

⇨ Mọi thì đi với kéo theo, tồn tại một thì đi với và, ví dụ

P(x) = “x là con cu”
Q(x) = “x > 20 cm”
R(x) = “x rất mạnh”
T(x) = “Mọi con cu đều > 20 cm”
Y(x) = “Tồn tại một con cu rất mạnh”
$T(x) = \forall x\big(P(x) \to Q(x)\big)$
$Y(x) = \exists x\big(P(x) \wedge R(x)\big)$

⇨ Nếu mọi và tồn tại một lồng vào nhau, thì đọc từ trái sang

$$\forall x \exists y P(x, y)$$

⇨ Đọc là “Với mọi x tồn tại một y sao cho P(x, y)”

$$\exists x \forall y P(x, y)$$

⇨ Đọc là “Tồn tại một x sao cho với mọi y, P(x, y)”

13. Quy Luật Suy Ra?

⇨ Giả sử ta có lập luận sau

⇨ Vì p_1, p_2, … đúng, do đó q đúng

⇨ Kí hiệu lập luận này như sau

$$\begin{array}{l} p_1 \\ p_2 \\ \hline \cdots \\ \therefore q \end{array}$$

⇨ Dấu ∴ gọi là dấu do đó, p_1, p_2, … gọi là các tiền đề (Premise)

⇨ Phương pháp khẳng định (Modus Ponens)

⇨ p đã xảy ra, mà p xảy ra thì q phải xảy ra, do đó q đã xảy ra

$$\begin{array}{l} p \\ p \to q \\ \hline \therefore q \end{array}$$

⇨ Phương pháp phủ nhận (Modus Tollens)

⇨ p xảy ra thì q phải xảy ra, mà q không xảy ra nên p không thể xảy ra

$$\begin{array}{l} p \to q \\ \bar{q} \\ \hline \therefore \bar{p} \end{array}$$

⇨ Phương pháp tam đoạn luận giả định (Hypothetical Syllogism)

⇨ Nếu p xảy ra thì q phải xảy ra, mà nếu q xảy ra thì r phải xảy ra, do đó nếu p xảy ra thì chắc chắn r phải xảy ra

$$\begin{array}{l} p \to q \\ q \to r \\ \hline \therefore p \to r \end{array}$$

⇨ Phương pháp tam đoạn luận tuyển (Disjunctive Syllogism)

⇨ p hoặc q đã xảy ra, mà p không thể xảy ra nên chỉ có q xảy ra

$$\begin{array}{l} p \vee q \\ \bar{p} \\ \hline \therefore q \end{array}$$

⇨ Phương pháp phủ định (Negation)

⇨ Nếu p xảy ra thì q phải vừa xảy ra vừa không xảy ra, do đó p phải không xảy ra

$$\begin{array}{l} p \to q \\ p \to \bar{q} \\ \hline \therefore \bar{p} \end{array}$$

⇨ Phương pháp phân tích trường hợp (Case Analysis)

⇨ Nếu p xảy ra thì r phải xảy ra, tương tự nếu q xảy ra thì r cũng phải xảy ra, do đó nếu chỉ cần p hoặc q xảy ra thì r sẽ xảy ra

$$\begin{array}{l} p \to r \\ q \to r \\ p \vee q \\ \hline \therefore r \end{array}$$

⇨ Phương pháp xây dựng song đề (Constructive Dilemma)

⇨ Nếu p xảy ra thì q phải xảy ra, tương tự nếu r xảy ra thì s phải xảy ra, do đó nếu 1 trong 2 p hoặc r xảy ra thì 1 trong 2 q hoặc s sẽ xảy ra

$$\begin{array}{l} p \to q \\ r \to s \\ p \vee r \\ \hline \therefore q \vee s \end{array}$$

⇨ Phương pháp cộng (Addition)

$$\begin{array}{l} p \\ \hline \therefore p \vee q \end{array}$$

⇨ Phương pháp rút gọn (Simplification)

$$\begin{array}{l} p \wedge q \\ \hline \therefore p \end{array}$$

⇨ Phương pháp kết hợp (Conjunction)

$$\begin{array}{l} p \\ q \\ \hline \therefore p \wedge q \end{array}$$

⇨ Phương pháp phân giải (Resolution)

$$\begin{array}{l} p \vee q \\ \bar{p} \vee r \\ \hline \therefore q \vee r \end{array}$$

⇨ Phương pháp cụ thể hóa phổ quát (Universal Instantiation)

⇨ c là một giá trị cụ thể nào đấy

$$\begin{array}{l} \forall x P(x) \\ \hline \therefore P(c) \end{array}$$

⇨ Phương pháp tổng quát hóa phổ quát (Universal Generalization)

$$\begin{array}{l} P(c), c\ random \\ \hline \therefore \forall x P(x) \end{array}$$

⇨ Phương pháp cụ thể hóa tồn tại (Existential Instantiation)

$$\begin{array}{l} \exists x P(x) \\ \hline \therefore P(c), some\ c \end{array}$$

⇨ Phương pháp tổng quát hóa tồn tại (Existential Generalization)

$$\begin{array}{l} P(c), some\ c \\ \hline \therefore \exists x P(x) \end{array}$$

⇨ Về bản chất, gần như mọi giả thiết đều có thể đưa về dạng p suy ra q rồi ta sử dụng tính chất bắc cầu để suy luận

⇨ Một số phương pháp khác

⇨ Nếu p xảy ra thì q phải xảy ra, tương tự nếu r xảy ra thì s phải xảy ra, mà q và s không thể đồng thời xảy ra, do đó p và r cũng không thể đồng thời xảy ra

$$\begin{array}{l} p \to q \\ r \to s \\ \overline{q \wedge s} \\ \hline \therefore \overline{p \wedge r} \end{array}$$

⇨ Nếu p xảy ra thì xảy ra thì q chắc chắn xảy ra, mà nếu r xảy ra thì chắc chắn q không xảy ra, do đó nếu p xảy ra thì r không được xảy ra

$$\begin{array}{l} p \to q \\ r \to \bar{q} \\ \hline \therefore p \to \bar{r} \end{array}$$

⇨ Nếu p và q cùng xảy ra trên 1 người nào đó thì đã có người dính p và có người dính q

$$\begin{array}{l} \exists x\big(P(x) \wedge Q(x)\big) \\ \hline \therefore \exists x P(x) \wedge \exists x Q(x) \end{array}$$

14. Ngụy Biện (Fallacy)?

⇨ Sử dụng quy luật suy ra trên các mệnh đề không phải hằng đúng, ví dụ

> "Nếu làm đúng 100 câu thì 10 điểm, và bạn được 10 điểm, do đó bạn làm đúng 100 câu"

⇨ Rõ ràng suy luận trên sai vì có thể làm đúng 90 câu là được 10 điểm rồi

⇨ Để kiểm tra 1 suy luận đúng hay sai thì biến nó về biểu thức Boolean rồi lập biểu đồ Karnaugh, nếu Full 1 thì đúng, ngược lại sai, cách dễ hơn là giả sử kết luận là sai, sau đó thế ngược lên trên, làm sao cho mọi giả thuyết đều đúng, thì khi này suy luận này là sai, nhưng nếu ta không thể làm cho mọi giả thuyết đúng hết, thì suy luận này là đúng

15. Phép Chứng Minh?

⇨ Tiên đề (Axiom) là mệnh đề chúng ta cho là đúng, là cái sơ khai nhất, ví dụ 1 + 1 = 2

⇨ Định lý (Theorem) là mệnh đề có thể được chứng minh thành đúng

⇨ Bổ đề (Lemma) là định lý phụ dùng góp phần chứng minh định lý quan trọng hơn

⇨ Hệ quả (Corollary) là 1 định lý khác có thể suy ra trực tiếp từ định lý vừa được chứng minh

⇨ Phỏng đoán (Conjecture) là 1 mệnh đề được dự đoán là đúng, và sẽ trở thành định lý 1 khi được chứng minh

⇨ Có 4 cách chứng minh 1 định lý có dạng p suy ra q

⇨ Chứng minh trực tiếp, giả sử p đúng, rồi ta bằng các bước suy luận để ra q

⇨ Chứng minh phản đảo, hay chứng minh gián tiếp, mệnh đề phản đảo đúng thì mệnh đề đầu cũng đúng, nghĩa là ta giả sử q' đúng, rồi từ q' ta bằng các bước suy luận để ra p', q' suy ra p' đúng thì p suy ra q đúng

⇨ Chứng minh phản chứng, hay chứng minh mâu thuẫn, bạn sẽ giả thiết là p và q' là đúng, từ 2 giả thiết này bạn suy luận ra 1 giả thiết khác là r, và cũng từ 2 giả thiết này để suy luận ra r', và do đó ta có biểu thức sau phải đúng

$$(p \wedge \bar{q}) \to (r \wedge \bar{r}) \equiv (p \wedge \bar{q}) \to F \equiv \overline{p \wedge \bar{q}} \equiv \bar{p} \vee q \equiv p \to q$$

⇨ Ví dụ, chứng minh nếu quan hệ R có tính bắc cầu, thì R^2 cũng phải có tính bắc cầu, đặt p là mệnh đề "R có tính bắc cầu" và q là mệnh đề "R^2 có tính bắc cầu"

$$p\ (1)$$
$$\bar{q}\ (2)$$
$$(a,b),(b,c) \in R^2\ (3)$$
$$(a,c) \notin R^2\ (4, from\ 2)$$
$$\exists e,(a,e),(e,b) \in R\ (5, from\ 3)$$
$$(e,c) \notin R\ (6, from\ 4{,}5)$$
$$\exists k,(b,k),(k,c) \in R\ (7, from\ 3)$$
$$(b,c) \in R\ (8, from\ 1{,}7)$$
$$(e,c) \in R\ (9, from\ 1{,}5{,}8)$$
$$\therefore p \to q$$

⇨ Ở đây, dòng (6) với (9) tạo thành cặp r và r'

⇨ Chứng minh quy nạp, chứng minh nó đúng với x = 1, rồi nó đúng với x = x + 1

16. Đồng Dư (Congruence Modulo)?

⇨ Xét mệnh đề

$a \equiv b(mod\ m)$

⇨ Mệnh đề này tương đương

"Phần dư khi lấy a chia m bằng phần dư khi lấy b chia m"

⇨ Xét mệnh đề

$a|b$

⇨ Mệnh đề này tương đương

"b chia hết cho a"
"a là 1 ước của b"

⇨ Xét mệnh đề

$a \nmid b$

⇨ Mệnh đề này tương đương

"b không chia hết cho a"
"a không phải 1 ước của b"

17. Đối Ngẫu (Duality)?

⇨ Cho biểu thức Boolean p chỉ gồm các toán tử AND, OR, NOT, đối ngẫu của p là biểu thức p* sao cho p* đạt được bằng cách thế mỗi phép AND trong p thành OR, và mỗi phép OR trong p thành AND, toán tử NOT không động đến, nếu trong p có hằng đúng, kí hiệu T, thì chuyển nó thành hằng sai, kí hiệu F, và ngược lại

⇨ Ví dụ

$p = a \wedge \left(\overline{b \wedge c} \vee d\right) \vee \bar{e} \vee T \wedge F \Rightarrow p^* = a \vee \left(\overline{b \vee c} \wedge d\right) \wedge \bar{e} \wedge T \vee F$

⇨ Biểu đồ Karnaugh của p* = biểu đồ Karnaugh của p nhưng thay 1 thành 0 và 0 thành 1, kể cả mã Gray

⇨ Ví dụ

⇨ Biểu đồ Karnaugh của p

	00	01	11	10
00	1	1	1	0
01	1	0	0	1
11	0	1	0	1
10	1	0	1	0

⇨ Biểu đồ Karnaugh của p*

	11	10	00	01

11	0	0	0	1
10	0	1	1	0
00	1	0	1	0
01	0	1	0	1

⇨ Do đó, nếu p và q tương đương, thì p* và q* cũng tương đương

Computer Architecture – Cấu Trúc Máy Tính:

1. RAM?

⇨ Mỗi ô nhớ trong RAM sẽ gồm 1 Byte, mỗi ô nhớ sẽ có địa chỉ vật lý riêng = số nhị phân, tăng dần từ đáy đến đỉnh RAM

2. Mô Hình Von Neumann (Princeton)?

⇨ Chia thành 4 hệ thống con là Input/Output, bộ điều khiển, ALU, bộ nhớ

⇨ Instruction và dữ liệu cùng được lưu trong RAM

⇨ Instruction được thực thi lần lượt

⇨ 1 chu trình Clock bao gồm Fetch, Decode, Execute

3. Mô Hình Harvard?

⇨ Y chang Von Neumann nhưng Instruction được lưu tại bộ nhớ riêng so với dữ liệu, do đó cần 2 bộ dây dẫn từ vi xử lí sang 2 bộ nhớ

⇨ Bộ nhớ lưu Instruction không bay hơi (ROM) còn lưu dữ liệu thì bay hơi (RAM)

⇨ Do không dùng chung bộ nhớ nên phải chia tỉ lệ giữa 2 bộ nhớ sao cho hợp lí, nếu không sẽ có bộ nhớ trống mà chẳng dùng để làm gì

4. Vi Xử Lí 8086 (iAPX 86, Intel 8086)?

⇨ Là 1 kiến trúc, bao gồm các phần sau

⇨ Các Register 16 Bit là CS, DS, ES, SS, IP, DI, SI, BP, SP, AX, BX, CX, DX

⇨ AX, BX, CX, DX mỗi cái chia thành 2 Register 8 Bit, ví dụ AX = AH (MSB) và AL (LSB), tương tự BX = BH và BL, …

⇨ Các Flag 1 Bit như Directional Flag, Zero Flag, …, giá trị của nó chỉ bị thay đổi khi có Instruction can thiệp, không thay đổi khi Process kết thúc

⇨ RAM có kích thước 1 MB

⇨ ALU

⇨ Địa chỉ của Instruction tiếp theo sẽ được thực hiện trong RAM luôn được xác định = CS * 16 + IP trong hệ thập phân

⇨ Cứ mỗi xung Clock, 1 Instruction tiếp theo sẽ được thực hiện, đồng thời IP tăng thêm 1

5. Vi Xử Lí x86 (80x86, ix86)?

⇨ Vi xử lí 586 còn gọi là Pentium, 686 là Pentium 2, 786 là Pentium 3, …

⇨ x = 1, 2 thì vi xử lí 16 Bit

⇨ x từ 3 trở lên thì vi xử lí 32 Bit

6. Vi Xử Lí x86 – 64 (x64, Intel 64, AMD64)?

7. Máy 32 Bit Và 64 Bit?

⇨ Máy x bit thì trong 1 xung Clock, nó chỉ thực hiện 1 phép tính với toán tử có kích thước x bit, hay nói cách khác, chỉ làm việc với Register có kích thước x bit

8. Cấu Tạo Máy Tính?

⇨ Bao gồm nhiều vi xử lí

⇨ Mỗi vi xử lí bao gồm nhiều Core

⇨ Mỗi Core sẽ bao gồm 1 bộ nhớ đệm L1 và 1 bộ nhớ đệm L2, L1 lại được chia thành 2 phần tương ứng với dữ liệu và Instruction, toàn bộ Core dùng chung thêm bộ nhớ đệm L3, tốc độ L1 > L2 > L3
⇨ Mỗi Core sẽ có bộ Register và ALU của riêng nó

9. Context Switch?

⇨ Thread là 1 phần trong Process, ví dụ 1 Process gồm 1 Thread là chạy Video và 1 Thread là hiển thị phụ đề
⇨ Context Switch là việc chuyển đổi qua lại giữa các Process hoặc các Thread, = cách lưu trạng thái hiện tại của Process hoặc Thread vào bộ nhớ, rồi chuyển sang Process hoặc Thread khác, sau đó chuyển lại Process hoặc Thread ban đầu = bộ nhớ, thứ cần lưu sẽ là giá trị hiện tại của Register, …, ví dụ cần lưu Register CS và IP để khi quay lại biết chỗ của Instruction cần thực thi
⇨ Việc chuyển đổi sẽ xảy ra khi Process hoặc Thread hiện tại đang gửi tín hiệu tới đĩa cứng, hoặc ngược lại, vì thời gian chờ lâu, tín hiệu Interrupt sẽ được bắn cho hệ điều hành để nó lưu Process hoặc Thread hiện tại và chuyển sang Process hoặc Thread khác, hoặc khi người dùng chuyển Tab, …

10. RISC (Reduced Instruction Set Computer) Và CISC (Complex Instruction Set Computer)?

⇨ RISC có Instruction chỉ làm 1 việc duy nhất, ví dụ Copy dữ liệu từ vi xử lí sang bộ nhớ
⇨ CISC có Instruction làm nhiều việc liên tục, ví dụ Copy xong rồi xóa

11. SISD (Single Instruction Single Data), MIMD (Multiple Instruction Multiple Data), MISD, SIMD?

⇨ SISD là 1 Instruction hoạt động trên 1 Data, như mô hình Von Neumann
⇨ SIMD là 1 Instruction được phân thân rồi hoạt động trên nhiều Data
⇨ MISD là 1 Data phân thân rồi được nhiều Instruction hoạt động trên
⇨ MIMD là nhiều Instruction hoạt động trên nhiều Data cùng lúc, 1 Instruction 1 Data khác nhau

12. DSP (Digital Signal Processor)?

⇨ Vi xử lí phục vụ chính cho việc xử lí âm thanh, Video, …, nhiều Instruction chạy song song
⇨ Sử dụng mô hình Harvard

13. SHARC (Super Harvard Architecture Single Chip Computer)?

⇨ Là 1 DSP cực mạnh

14. Vi Điều Khiển (Micro Controller)?

⇨ Là 1 IC mà trong nó có chứa cả vi xử lí cùng với các thiết bị ngoại vi
⇨ Các Loại Micro Controller
⇨ ARM (Advanced RISC Machines), ARMx với x < 9 sử dụng mô hình Von Neumann, >= 9 sử dụng Harvard
⇨ PIC (Peripheral Interface Controller), sử dụng mô hình Harvard

15. DASD (Direct Access Storage Device) và SASD (Sequential Access Storage Device)?

⇨ DASD là bộ nhớ mà chỉ cần có địa chỉ là truy xuất ngay lập tức, gồm đĩa cứng, đĩa mềm, đĩa CD
⇨ SASD là bộ nhớ mà thời gian truy xuất phụ thuộc địa chỉ, gồm băng từ

16. I/O Controller?

⇨ Nhiệm vụ là điều khiển việc đưa dữ liệu từ RAM vào bộ nhớ đệm để ghi vào bộ nhớ ngoài, do bộ nhớ ngoài ghi rất chậm so với RAM, đồng thời cho phép vi xử lí làm việc khác khi đang ghi
⇨ Gửi tín hiệu Interrupt cho vi xử lí khi ghi xong

17. AMD (Advanced Micro Devices)?
⇨ Là 1 công ty

18. Paged Memory?
⇨ Hệ điều hành chịu trách nhiệm tạo ra Paged Memory, không phải vi xử lí
⇨ Giả sử trong Code của Process ta có ghi địa chỉ là ABCDEF01, đây là địa chỉ 32 Bit, thì bản chất nó đéo phải địa chỉ vật lý trong RAM mà là địa chỉ ảo
⇨ Địa chỉ này sẽ được ánh xạ thành địa chỉ vật lý trong RAM thông qua Page Table
⇨ Mỗi Process trước khi được tải vào RAM để chạy, sẽ được cấp phát cho 1 Page Table riêng, lưu tại RAM, ở khu vực gần Kernel
⇨ CR3 là Register có giá trị là địa chỉ của Page Table ứng với Process hiện tại, được thay đổi khi xảy ra Context Switch
⇨ Giả sử địa chỉ vật lý trong RAM có giá trị từ 00000000 đến FFFFFFFF, như vậy RAM có kích thước 4 GB
⇨ Hệ điều hành chia địa chỉ ảo, giả sử thành 4 phần, nghĩa là đây là 3 Level Paged Memory, tương tự nếu k phần thì k – 1 Level, thành AB, CD, E, F01
⇨ Page Table của Process trường hợp này bản chất gồm rất nhiều Page Table con, chia làm 3 Level
⇨ AB ứng với Index của Level 1 Page Table, CD ứng với Index của Level 2, E thì Level 3, còn F01 thì là Offset
⇨ Dựa vào CR3 cùng 3 Index và Offset để xác định địa chỉ vật lý trong RAM
⇨ Ý tưởng như sau
⇨ Đầu tiên Copy nguyên thanh RAM vào não bạn, tức là bạn đang tưởng tượng nó
⇨ Level 1 Page Table, chia thanh RAM trong não bạn thành 256 phần y chang nhau liên tiếp (do 2 kí tự thập lục phân ứng với 256 Index), như vậy mỗi phần sẽ có kích thước 16 MB
⇨ Level 2 Page Table, chia mỗi phần ở trên thành 256 phần nhỏ nữa (do 2 kí tự thập lục phân ứng với 256 Index), như vậy mỗi phần sẽ có kích thước 64 KB, phần nào ở trên hoàn toàn không có dữ liệu thì không chia, bỏ qua
⇨ Level 3 Page Table, chia mỗi phần ở trên thành 16 phần nhỏ hơn nữa (do 1 kí tự thập lục phân ứng với 16 Index), như vậy mỗi phần sẽ có kích thước 4 KB, phần nào ở trên hoàn toàn không có dữ liệu thì không chia, bỏ qua
⇨ Tóm lại, chia thanh RAM trong não bạn thành các phần liên tiếp y chang nhau kích thước 4 KB, phần nào không chứa dữ liệu thì bỏ qua, mỗi phần sẽ được ánh xạ vào 1 địa chỉ vật lý trong RAM, địa chỉ này được lưu trong Page Table Level cao nhất, để truy cập 1 địa chỉ cụ thể trong 1 Page 4 KB, thì dùng Offset, dễ thấy Offset gồm 3 kí tự thập lục phân, tương ứng với 4096 Index = 4 KB
⇨ Tổng kích thước của Page Table mỗi Process khi Level đủ cao thông thường sẽ = (số Page của toàn bộ thanh RAM trong não, mà có dữ liệu của Process) Byte, ví dụ RAM 8 GB, kích thước Page 4 KB, Process sử dụng 1 GB dữ liệu, thì kích thước Page Table xấp xỉ (1 GB / 4 KB) Byte = 256 KB, khá ít nên có thể lưu nhiều Page Table của nhiều Process
⇨ Các Pointer trong ngôn ngữ lập trình luôn trả về địa chỉ ảo

19. Descriptor Table?

- ⇨ Các vi xử lí x86 ban đầu khi khởi chạy sẽ chạy y chang 8086, tức là địa chỉ ảo = địa chỉ thật, chỉ được truy cập 1 MB đầu tiên của RAM
- ⇨ Sau đó, chúng sẽ chuyển sang chế độ bảo vệ, có thể truy cập toàn bộ RAM bằng cácasckhi 1 Process chạy, nó sẽ tạo 1 Descriptor Table ở đâu đó trong RAM, sẽ có Register lưu địa chỉ của Table này
- ⇨ Địa chỉ của Instruction tiếp theo sẽ không còn = CS * 16 + IP nữa, thứ CS trả về chính là 1 Index trong Desciptor Table ứng với Process này, giá trị lưu tại Index này sẽ bao gồm các thông tin như vị trí của phần nhớ lưu toàn bộ Instruction của Process trong địa chỉ ảo, kích thước phần nhớ, …, và địa chỉ của Instruction tiếp theo sẽ dựa vào thông tin này kèm theo IP
- ⇨ Tương tự với các Register DS, SS, FS, GS, ES

Encoding – Mã Hóa:

1. Cách Mã Hóa Các Kí Tự Chữ Cái Thành Vector?

- ⇨ Giả sử có 26 chữ cái, thì mỗi chữ cái tương ứng với 1 One Hot Vector, vâng, đó là Vector mà mọi phần tử đều = 0 ngoại trừ phần tử có vị trí là số thứ tự của chữ cái có giá trị = 1
- ⇨ Ví dụ
- ⇨ Mã hóa chữ b khi Input chỉ có thể là a, b, c, d, e

$$'b' = (0 \quad 1 \quad 0 \quad 0 \quad 0)$$

2. Cách Mã Hóa Vị Trí Của Chữ Trong Văn Bản (Positional Encoding)?

- ⇨ Giả sử từ “foo” trong câu được biểu diễn = 1 Vector A có n chiều, tuy nhiên Vector này chưa có thông tin về vị trí của “foo”, do đó ta cần + A với 1 Vector khác mang thông tin vị trí của “foo”, gọi là B, cũng có n chiều
- ⇨ Tưởng tượng bạn có 1 cái đồng hồ kim, dễ thấy trong vòng 12 giờ, không có 2 thời điểm nào đồng hồ có trạng thái giống nhau
- ⇨ Ta có thể sử dụng vị trí của kim đồng hồ để mã hóa vị trí của mỗi từ trong câu mà không sợ bị lặp lại, chỉ cần chu kì đủ lớn
- ⇨ Như vậy, giả sử chiếc đồng hồ mã hóa có n / 2 kim, kim thứ k chậm hơn kim thứ k – 1 khoảng m lần, trạng thái ban đầu là tất cả các kim trùng nhau và chĩa lên trên, hướng quay cùng chiều kim đồng hồ bình thường, để mô tả vị trí kim, ta sẽ dùng hoành độ và tung độ của nó, lưu ý độ dài tất cả các kim = 1, và thay vì là hàm theo thời gian, vị trí kim sẽ là hàm theo vị trí của từ, khi đó B sẽ có dạng như sau

$$B = (x_0(p) \quad y_0(p) \quad x_1(p) \quad y_1(p) \quad x_2(p) \quad y_2(p) \quad ...)$$

- ⇨ p là vị trí của “foo” trong câu, có giá trị là 0, 1, 2, …
- ⇨ x_n, y_n lần lượt là hàm trả về giá trị hoành độ và tung độ của kim thứ n tại thời điểm p
- ⇨ Dễ thấy, từ tất cả các điều kiện trên, x và y chỉ có thể là hàm Sin và Cos, coi chu kì của kim thứ 0 là 2π, ta có

$$B = \left(sin(p) \quad cos(p) \quad sin\left(\frac{1}{m}p\right) \quad cos\left(\frac{1}{m}p\right) \quad sin\left(\frac{1}{m^2}p\right) \quad cos\left(\frac{1}{m^2}p\right) \quad ...\right)$$

- ⇨ Với

$$m = 10000^{\frac{2}{n}}$$

- ⇨ Dễ thấy, đi từ phần tử đầu đến cuối của B, chu kì tăng theo cấp số nhân, từ 2π đến 20000π
- ⇨ Lợi ích khi sử dụng hàm Sin và Cos
- ⇨ Giả sử B_1 và B_2 lần lượt là Vector mã hóa vị trí của 2 từ "foo" và "bar" trong câu, giả sử foo ở vị trí thứ p và "bar" ở vị trí thứ p + t
- ⇨ Dễ thấy dù p có giá trị gì đi chăng nữa thì khoảng cách giữa B_1 và B_2 luôn không đổi, như vậy ta vô tình mã hóa được vị trí tương đối của 2 từ mà chẳng cần làm gì
- ⇨ Ví dụ
- ⇨ Khoảng cách của từ thứ 1 đến từ thứ 5 = khoảng cách của từ thứ 10 đến từ thứ 14
- ⇨ Chứng minh
- ⇨ Ta có

$$|B_2 - B_1| = |(sin(p+t) - sin(p) \quad cos(p+t) - cos(p) \quad \ldots)| =$$
$$\sqrt{|(sin(p+t) - sin(p) \quad cos(p+t) - cos(p))|^2 + \cdots} =$$
$$\sqrt{\left|\begin{pmatrix} sin(p+t) \\ cos(p+t) \end{pmatrix} - \begin{pmatrix} sin(p) \\ cos(p) \end{pmatrix}\right|^2 + \cdots} = \sqrt{|A_2 - A_1|^2 + \cdots}$$

- ⇨ Dễ thấy khoảng cách giữa A_2 và A_1 không phụ thuộc vào p mà chỉ phụ thuộc vào t, vì bản chất quay A_1 t Radian ngược chiều kim đồng hồ là được A_2, do đó khoảng cách giữa A_2 và A_1 chính = độ dài đáy của tam giác cân có cạnh bên = 1 và góc ở đỉnh = t Radian
- ⇨ Như vậy, dễ thấy khoảng cách giữa B_2 và B_1 không phụ thuộc p
- ⇨ Đồng thời, người ta cũng thấy rằng khoảng cách giữa B_2 và B_1 càng lớn nếu vị trí của "foo" và "bar" càng xa nhau, hợp lí

Loss Function – Hàm Thất Thoát:

1. MSE (Mean Squared Error)?
- ⇨ Có 2 Vector A và B có cùng số phần tử, A là Vector có giá trị đúng, còn B là dự đoán, khi đó lấy B – A, bình phương các phần tử rồi lấy giá trị trung bình là ra MSE
2. Độ Ngạc Nhiên (Surprise) Là Gì?
- ⇨ Giả sử tỉ lệ xuất hiện của 1 giá trị rất nhỏ, nhưng thế đéo nào ta lại Sample được giá trị ấy, khi đó ta ngạc nhiên vãi lồn, nên độ ngạc nhiên lớn, nhưng nếu tỉ lệ xuất hiện của giá trị đó rất lớn, thì bạn đéo ngạc nhiên, độ ngạc nhiên nhỏ
- ⇨ Như vậy dễ thấy độ ngạc nhiên tỉ lệ nghịch với xác suất, nên ta có công thức tính độ ngạc nhiên khi Sample được giá trị x như sau

$$surprise = ln\left(\frac{1}{P(X=x)}\right)$$

- ⇨ Ở đây ta không lấy luôn nghịch đảo của xác suất vì ta cần khi xác suất = 1 thì độ ngạc nhiên phải = 0
3. Entropy Là Gì?
- ⇨ Là kỳ vọng vọng của độ ngạc nhiên khi Sample rất nhiều giá trị
- ⇨ Entropy nghĩa là độ hỗn loạn, nên nếu phân phối có Variance lớn thì Entropy sẽ lớn, điều này có nghĩa là độ ngạc nhiên trung bình sẽ lớn nhất khi phân phối đều

nhất, điều này đúng, vì thử nghĩ xem, nếu phân phối chỉ dồn vào 1 giá trị, tỉ lệ xuất hiện của nó là 100%, thì khi Sample ta đéo ngạc nhiên chút nào vì lúc nào cũng ra giá trị đó, mà độ ngạc nhiên tỉ lệ thuận với Entropy nên Entropy cũng nhỏ

⇨ Dễ dàng tìm ra được công thức Entropy của 1 phân phối rời rạc

$$H(X) = E[surprise] = \sum_x ln\left(\frac{1}{P(X=x)}\right) P(X=x) = -\sum_x P(X=x) ln\big(P(X=x)\big)$$

⇨ X là biến ngẫu nhiên

⇨ Phân phối liên tục không có Entropy vì nếu tính Entropy thì nó là ∞, thay vào đó là Entropy vi phân có công thức sau

$$h(X) = -\int_{-\infty}^{\infty} f(x) ln\big(f(x)\big)\, dx$$

⇨ X là biến ngẫu nhiên có PDF là hàm f(x)

4. Cross Entropy là gì?

⇨ Gần giống với Entropy nhưng nó liên quan đến 2 phân phối, do đó gọi là Cross

⇨ Công thức Cross Entropy cho 2 phân phối rời rạc

$$H(A,B) = -\sum_x P(A=x) ln\big(P(B=x)\big)$$

⇨ A là biến ngẫu nhiên của phân phối gốc

⇨ B là biến ngẫu nhiên của phân phối dự đoán

⇨ Cũng giống Entropy, Phân phối liên tục không có Cross Entropy, thay vào đó là Cross Entropy vi phân

$$h(A,B) = -\int_{-\infty}^{\infty} f(x) ln\big(g(x)\big)\, dx$$

⇨ A là biến ngẫu nhiên của phân phối gốc có PDF là hàm f(x)

⇨ B là biến ngẫu nhiên của phân phối dự đoán có PDF là hàm g(x)

⇨ Ta có bất phương trình sau

$$h(A,B) \geq h(A)$$

⇨ Nghĩa là Cross Entropy vi phân không bao giờ nhỏ hơn Entropy vi phân

⇨ Chứng minh

$$\int_{-\infty}^{\infty} f(x) ln\left(\frac{g(x)}{f(x)}\right) dx \leq \int_{-\infty}^{\infty} f(x)\left(\frac{g(x)}{f(x)} - 1\right) dx = \int_{-\infty}^{\infty} f(x)\, dx - \int_{-\infty}^{\infty} g(x)\, dx =$$
$$1 - 1 = 0 \Rightarrow \int_{-\infty}^{\infty} f(x) ln\left(\frac{g(x)}{f(x)}\right) dx \leq 0 \Rightarrow$$
$$\int_{-\infty}^{\infty} f(x) ln\big(g(x)\big)\, dx - \int_{-\infty}^{\infty} f(x) ln\big(f(x)\big)\, dx \leq 0 \Rightarrow$$
$$-\int_{-\infty}^{\infty} f(x) ln\big(g(x)\big)\, dx \geq -\int_{-\infty}^{\infty} f(x) ln\big(f(x)\big)\, dx \Rightarrow h(A,B) \geq h(A)$$

⇨ Tương tự dễ dàng chứng minh được Cross Entropy cũng không bao giờ nhỏ hơn Entropy

5. Cách Tính Khoảng Cách Hay Sự Khác Biệt Giữa 2 Phân Phối = Phân Kỳ KL (Kullback Leibler Divergence)?

⇨ Bản chất là = Cross Entropy – Entropy, ở đây nói chung cho cả vi phân

⇨ Danh xưng khoảng cách là vì nó không bao giờ nhỏ hơn 0 do Cross Entropy không bao giờ nhỏ hơn Entropy, ngoài ra là vì chỉ khi 2 phân phối = nhau hoàn toàn thì nó mới = 0, và càng lớn khi 2 phân phối càng khác nhau, do đó nó không giao hoán cũng là chuyện bình thường, không giao hoán ở đây nghĩa khi thay đổi vai trò của 2 phân phối thì phân kỳ KL sẽ thay đổi

⇨ Bản chất khi bạn cố gắng giảm giá trị Cross Entropy Loss là bạn đang làm giảm phân kỳ KL hay nói cách khác là khoảng cách giữa phân phối đúng và phân phối dự đoán khi sử dụng hàm Softmax của bạn

⇨ Khi Train Model để dạng phân loại, thì nên sử dụng Cross Entropy Loss vì nó rất dễ tính, do phân phối đúng hay nhãn là One Hot Vector, nên ta chỉ cần tính – ln(<Giá Trị Dự Đoán Có Vị Trí = Vị Trí Của 1 Trong One Hot Vector>)

6. Tại Sao Lại Sử Dụng Cross Entropy Thay Cho MSE?

⇨ Trong trường hợp Train Model dạng phân loại, thì ta thường sử dụng Softmax cho Layer cuối cùng, nên giá trị của dự đoán luôn nằm trong khoảng (0, 1), mà đạo hàm của MSE lại tuyến tính nên Gradient sẽ nhỏ và xêm xêm nhau, trong khi khi sử dụng hàm ln thì, nếu giá trị dự đoán có vị trí của 1 trong One Hot Vector gần 0, nói cách khác là nó dự đoán sai bét, thì đạo hàm của ln tại số gần 0 lại cực lớn, do đó Gradient sẽ lớn, hợp lý vì nó cần phải sửa sai nhanh

7. Multi Class Khác Đéo Gì Multi Label?

⇨ Phân loại kiểu Multi Class nghĩa là Layer cuối cùng sử dụng hàm Softmax, nhãn là One Hot Vector, sử dụng Cross Entropy Loss, không sử dụng Binary Cross Entropy Loss vì hiệu suất tính toán

⇨ Phân loại kiểu Multi Label nghĩa là Layer cuối cùng sử dụng hàm Sigmoid, nhãn là Binary Vector, có thể có nhiều số 1, sử dụng Binary Cross Entropy Loss

8. Binary Cross Entropy Là Cái Lồn Gì?

⇨ Xét trường hợp phân loại 2 lớp, ví dụ “chó” và “mèo”, như vậy ta chỉ cần 1 Neuron ở Layer cuối cùng, trả về 1 là “chó”, 0 là “mèo”, để ý thấy khi nhãn = 0 thì Cross Entropy = 0, đéo cần biết là dự đoán = 1 sai vãi đái, do đó ta sử dụng Binary Cross Entropy có thể tích Loss trong trường hợp nhãn = 0

$$L = -\left(y_T ln(y_P) + (1 - y_T) ln(1 - y_P)\right)$$

⇨ y_T là giá trị dự đoán, thuộc (0, 1)

⇨ y_p là giá trị nhãn, = 0 hoặc 1

⇨ Trường hợp phân loại kiểu Multi Label thì tính Loss của mỗi Neuron trong lớp cuối cùng = Binary Cross Entropy rồi lấy giá trị trung bình, không được lấy Cross Entropy trong trường hợp này để đề phòng trường hợp đéo có nhãn nào = 1, nghĩa là tổng Cross Entropy = 0, đéo tính được Gradient

Activation Function – Hàm Kích Hoạt:

1. Hàm ReLU (Rectified Linear Unit)?

$$f(x) = \begin{cases} x, x \geq 0 \\ 0, x < 0 \end{cases}$$

2. Hàm Sigmoid?

⇨ Có Codomain là (–1, 1)

$$f(x) = \frac{1}{1+e^{-x}}$$

3. Hàm Softmax?

⇨ Hàm này áp dụng với Vector, e mũ mỗi phần tử của Vector rồi Scale sao cho tổng tất cả phần tử = 1

4. Hàm Log Softmax?

⇨ Bằng Softmax rồi lấy Logarithm tự nhiên của kết quả nhận được

5. Hàm GELU (Gaussian Error Linear Unit)?

⇨ Hàm này giống ReLU vãi nhưng là phiên bản mượt trên R

$$f(x) = x\Phi(x)$$

⇨ $\Phi(x)$ là CDF của phân phối chuẩn tắc

6. Hàm ELU (Exponential Linear Unit)?

⇨ Sự kết hợp giữa Sigmoid và ReLU

⇨ Có Codomain là $(-\alpha, \infty)$

$$f(x) = \begin{cases} x, x \geq 0 \\ \alpha(e^x - 1), x < 0 \end{cases}$$

⇨ Nếu α = 1 thì mượt trên R, khác 1 thì gãy khúc tại x = 0

7. Hàm LReLU (Leaky ReLU)?

⇨ Phiên bản nhánh trái có hệ góc > 0 của ReLU

$$f(x) = \begin{cases} x, x \geq 0 \\ ax, x < 0 \end{cases}$$

⇨ a là hệ số góc nhánh trái

8. Hàm RreLU(Randomized Leaky ReLU)?

⇨ Là LreLU nhưng trong quá trình Train, a sẽ liên tục biến đổi ngẫu nhiên bằng cách Sample từ 1 phân phối đều liên tục

⇨ Khi Test, a sẽ được cố định và = Mean của phân phối đều liên tục

Optimizer – Tối Ưu Hóa:

1. Optimizer Dùng Để Làm Gì?

⇨ Giả sự bộ dữ liệu để Train của bạn có hàng triệu phần tử thì bạn không thể nào Feed hết 1 lần vào mạng để Train được, mà phải bóc tách thành các Sample rồi Feed lần lượt vào mạng

⇨ Optimizer là những thuật toán giúp bạn cải thiện các thông số 1 cách tối ưu nhất khi chỉ được Feed Sample mà không phải Population

2. Epoch Là Gì?

⇨ Bạn cứ Feed Sample cho đến khi hết bộ dữ liệu thì gọi là 1 Epoch

3. Gradient Descent?

⇨ Là Feed 1 lần cả bộ dữ liệu rồi tính Gradient trung bình, rồi cải thiện thông số, rồi lặp lại

⇨ Mỗi lần Feed là 1 Epoch

4. SGD (Stochastic Gradient Descent)?

⇨ Là bóc tách bộ dữ liệu thành từng Sample rồi Feed, rồi tính Gradient trung bình, rồi cải thiện thông số, rồi lặp lại

⇨ Khoảng mấy lần Feed mới được 1 Epoch

⇨ Giá trị hội tụ của thông số khi sử dụng SGD = khi sử dụng Gradient Descent thuần túy, với cùng số lần Feed rất lớn

⇨ Nếu Sample có kích thước = 1, gọi là SGD, nếu > 1 thì gọi là Mini Batch SGD

5. Chứng Minh SGD Hội Tụ Y Chang Gradient Descent?

⇨ Gọi toàn bộ thông số trong mạng là Vector θ

⇨ Gọi mỗi dữ liệu Train là Vector $X_1, X_2, X_3, \ldots, X_n$, bao gồm cả Input và nhãn, n là kích thước bộ dữ liệu

⇨ Toàn bộ bước đi dùng để cải thiện thông số sau 1 lần Feed ứng với dữ liệu Train X_i là Vector $f(\theta, X_i)$

⇨ Khi sử dụng Gradient Descent, ta có

$$\theta_{new} = \theta_{old} + \frac{1}{n}\sum_{i=1}^{n} f(\theta_{old}, X_i)$$

⇨ Giá trị hội tụ của θ là nghiệm của phương trình

$$\theta = \theta + \frac{1}{n}\sum_{i=1}^{n} f(\theta, X_i) \Rightarrow \frac{1}{n}\sum_{i=1}^{n} f(\theta, X_i) = 0$$

⇨ Khi sử dụng SGD với kích thước Sample là m, ta có

$$\theta'_{new} = \theta'_{old} + \frac{1}{m}\sum_{i=1}^{m} f(\theta'_{old}, X_{random})$$

⇨ Cho là hội tụ khi θ' thay đổi không đáng kể, khi đó, θ' là nghiệm của phương trình

$$\theta' = \theta' + E\left[\frac{1}{m}\sum_{i=1}^{m} f(\theta', X_{random})\right] \Rightarrow E\left[\frac{1}{m}\sum_{i=1}^{m} f(\theta', X_{random})\right] = 0$$

⇨ Dễ thấy đây là kỳ vọng của trung bình 1 thuộc tính trong Sample, và ta đã biết, nó chính = giá trị trung bình của thuộc tính đó trên cả Population

$$E\left[\frac{1}{m}\sum_{i=1}^{m} f(\theta', X_{random})\right] = 0 \Rightarrow \frac{1}{n}\sum_{i=1}^{n} f(\theta', X_i) = 0 = \frac{1}{n}\sum_{i=1}^{n} f(\theta, X_i) \Rightarrow \theta' = \theta$$

⇨ Vậy giá trị hội tụ của SGD y chang Gradient Descent

6. EWMA (Exponentially Weighted Moving Average) Là Gì?

⇨ Giả sử đồ thị dữ liệu của bạn nó Zic Zac vãi lồn và bạn tạo ra phiên bản mượt của nó, thì sử dụng trọng số để lưu lại một phần giá trị trước, cụ thể như sau

$$\nu_{new} = \beta\nu_{old} + (1 - \beta)\theta_{new}$$

⇨ ν là EWMA của giá trị θ, là phiên bản mượt

⇨ β là trọng số, thông thường = 0.95

7. SGD Với Momentum?

⇨ Dễ thấy do sử dụng Sample nên mỗi bước đi để cải thiện thông số sẽ có kiểu Zic Zac, để làm mượt nó thì sử dụng EWMA cho Gradient, như vậy hướng đi sẽ không biến đổi đột ngột, do giữ lại 1 phần của hướng đi trước

⇨ Hội tụ của SGD với Momentum cũng y chang Gradient Descent, là vì khi sử dụng SGD với Momentum, kích thước Sample là m, ta có

$$\theta'_{new} = \theta'_{old} + (1 - \beta)\frac{1}{m}\sum_{i=1}^{m} f(\theta'_{old}, X_{random}) + \beta\frac{1}{m}\sum_{i=1}^{m} f(\theta'_{older}, X_{random}) =$$

⇨ Cho là hội tụ khi θ' thay đổi không đáng kể, khi đó, θ' là nghiệm của phương trình

$$\theta' = \theta' + E\left[(1 - \beta)\frac{1}{m}\sum_{i=1}^{m} f(\theta', X_{random})\right] + E\left[\beta\frac{1}{m}\sum_{i=1}^{m} f(\theta', X_{random})\right] =$$
$$\theta' + E\left[\frac{1}{m}\sum_{i=1}^{m} f(\theta', X_{random})\right] \Rightarrow E\left[\frac{1}{m}\sum_{i=1}^{m} f(\theta', X_{random})\right] = 0$$

⇨ Dễ thấy phương trình này ta đã giải rồi, nghiệm của nó cũng chính là nghiệm của phương trình hội tụ Gradient Descent

8. Adagrad (Adaptive Gradient Descent)?

⇨ Giống y chang Gradient Descent chỉ khác độ dài bước đi

⇨ Dễ thấy chiều dài của bước đi ở mọi nơi gần như nhau, điều này không tốt vì có thể bỏ lỡ điểm cực tiểu, do đó, càng bước đi nhiều, càng tiến tới cực tiểu thì ta càng phải giảm chiều dài bước đi lại

⇨ Gọi tổng bình phương chiều dài các Gradient thuần túy cho tới thời điểm hiện tại là A, ta chia độ dài bước đi cho $(A + \varepsilon)^{0.5}$, ε là 1 số rất nhỏ đảm bao mẫu khác 0

⇨ Bất lợi là A càng lớn thì độ dài bước đi càng nhỏ, đến 1 lúc nó nhỏ vãi lồn thì lại hội tụ chậm

9. RMS (Root Mean Square) Là Gì?

⇨ Là căn bậc 2 của trung bình tổng bình phương các phần tử trong tập hợp nào đó

⇨ Cho tập hợp $\{x_1, x_2, x_3, \ldots, x_n\}$, n là số phần tử, RMS của tập này là

$$x_{RMS} = \sqrt{\frac{1}{n}(x_1^2 + x_2^2 + x_3^2 + \cdots + x_n^2)}$$

10. RMSprop (RMS Propagation)?

⇨ Giống Adagrad nhưng thay vì tổng bình phương chiều dài các Gradient thuần túy cho tới thời điểm hiện tại ta dùng EWMA của bình phương chiều dài Gradient, điều này ngăn chặn việc chiều dài bước đi bị rút ngắn vãi lồn, mà thay vào đó khi bị rút ngắn đến mức nào đó sẽ được hoãn lại

11. Adadelta?

⇨ Giống y chang RMSprop chỉ khác không cần sử dụng Learning Rate mà thay vào đó là 1 biểu thức giống ở dưới mẫu, đổi EWMA của bình phương chiều dài Gradient thành EWMA của bình phương chênh lệch chiều dài bước đi giữa 2 lần bước liên tiếp

⇨ Gọi là Delta vì trong công thức có Δx là chênh lệch chiều dài bước đi giữa 2 lần bước liên tiếp

⇨ Mục đích của việc đổi Learning Rate sang cái này là để thỏa mãn đơn vị của bước đi và tham số phải giống nhau

12. Adam (Adaptive Moment Estimation)?

⇨ Là kết hợp của RMSprop và SGD với Momentum

Weight Initialization – Khởi Tạo Trọng Số:

1. Tại Sao Không Khởi Tạo Weight Giống Nhau?

⇨ Nếu làm như vậy, thì khi Train, Gradient tại mọi nơi trên cùng 1 Layer sẽ giống nhau, nghĩa là mãi mãi nó sẽ giống nhau, mà ta muốn mỗi Neuron học 1 thứ khác nhau nên phải khởi tạo Weight khác nhau

2. Mean Và Variance Của Weight Khi Khởi Tạo?

⇨ Giả sử Weight kết nối giữa Layer 1 và Layer 2, điều ta muốn là giá trị của mỗi Neuron đều có Variance = 1, nếu lớn quá, thì khi chúng ta áp dụng các hàm kích hoạt như Sigmoid, thì Gradient sẽ trở nên rất nhỏ

⇨ Để làm được điều này, coi mỗi Neuron ở Layer 1 có giá trị = 1, đây là trường hợp xấu nhất, Layer này có m Neuron, nên mỗi Neuron ở Layer 2 sẽ được m Weight gắn vào, như vậy dễ thấy giá trị của Neuron này = tổng m Weight gắn vào nên Variance của nó = 1 = m lần Variance của Weight, nên Variance của Weight phải = 1 / m

Brute Force AI:

1. Ý Tưởng?

⇨ Giả sử Input là 1 điểm trong không gian n chiều, và giá trị được gán cho nó là 1 số nào đó, ta có rất nhiều Input như vậy, bài toán đặt ra là nếu ta cho AI 1 Input mới hoàn toàn thì giá trị nó trả về sẽ là bao nhiêu, cách đơn giản nhất để tìm ra là sử dụng phép nội suy, Input mới càng gần Input đã có nào thì giá trị của nó càng nghiêng về Input đó

⇨ Như vậy nghịch đảo khoảng cách mũ k của Input mới tới tất cả các Input đã có sẽ là trọng số, trọng số này được chuẩn hóa sao cho tổng của chúng = 1, k là độ tương phản, càng lớn thì giá trị trả về càng chính xác, thông thường đặt = 20

2. Pseudo Code?

⇨ Đặt độ tương phản

```
k = 20
```

⇨ Tính nghịch đảo khoảng cách mũ k

```
d = 1 / (((x_train - input) ** 2).sum((1, 2)) + 1e-5) ** k
```

⇨ Chuẩn hóa tỉ lệ

```
p = d / d.sum()
```

⇨ Kết quả

```
output = (p * y_train).sum()
```

3. Tỉ Lệ Đoán Đúng MNIST?

⇨ 96%

MLP – Multi Layer Perceptron:

1. Cấu Trúc?

⇨ 1 Input Layer, 1 Output Layer và ít nhất 1 Hidden Layer

⇨ Feed Forward, truyền từ Input thẳng tới Output

CNN – Convolutional Neural Network:

1. Không Gian Ẩn (Latent Space)?

⇨ Giả sử bạn có tấm ảnh con chó với hàng triệu Pixel, nói cách khác, tấm ảnh này nằm trong 1 chiều không gian có hàng triệu chiều, dễ thấy 1 lượng lớn Pixel như vậy không thể Feed trực tiếp vào 1 MLP, thay vào đó, ta sẽ xử lí để giảm số chiều của nó lại, khi đó tấm ảnh mới nằm trong không gian gọi là không gian ẩn, là không gian ít chiều nhưng chứa đủ thông tin

2. Bản Chất Toán Tử Convolution?

⇨ 1 lớp Convolution tương đương 1 lớp Fully Connected với đa số Weight = 0

UNET:

1. Ý Tưởng?

⇨ Là 1 mạng chỉ toàn lớp Convolution, ban đầu cho ảnh xuống Latent Space = Convolution, sau đó lại đưa lên lại kích thước ban đầu = Transpose Convolution, lúc đưa lên thì nối thêm thông tin lúc đưa xuống

⇨ Phù hợp cho Image Segmentation

Neural ODE – Neural Differential Equation:

1. Ý Tưởng?

⇨ Giả sử bạn có 1 MLP, trong có có 1 dãy các Layer ở giữa với cùng số Neuron, hay nói cách khác cùng số chiều là D, trong dãy này, gọi trạng thái của Layer đầu tiên là S_0, cuối cùng là S_N, lưu ý S là Vector dọc, thì thay vì phải tạo ra hàng tá Layer giữa S_0 và S_N, thì ta thay tất cả = 1 phương trình vi phân duy nhất, phương

trình này nhận S_0 làm giá trị xuất phát tại thời điểm t_0, có tác dụng đi từ S_0 đến S_N tại thời điểm t_N

⇨ Đây là phương trình vi phân bậc 1, có dạng

$$\frac{dS}{dt} = f(S, \theta, t)$$

⇨ Hàm f là 1 MLP khác có Input Layer với số chiều D + 1, D chiều đầu tiên là dùng cho S, trạng thái hiện tại, chiều cuối cùng dành cho t, thời gian hiện tại, còn θ là tham số của MLP như Weight, Bias, Output Layer thì có số chiều là D, là tốc độ thay đổi của S hiện tại

⇨ Sử dụng MLP để cập nhật giá trị của S theo thời gian bằng phương pháp Euler với chênh lệch thời gian giữa mỗi bước đi là h có giá trị nhỏ, sau khi đạt được trạng thái S_N thì Feed nó vào phần mạng tiếp theo, tính Loss rồi lan truyền ngược, khi này ta chỉ thu được Gradient của θ và tham số trong các phần mạng khác, do đó t_0 và t_N sẽ cố định và do con người đặt

⇨ Giả sử ta muốn cho AI học luôn t_0 và t_N, đồng thời khắc phục tình trạng quá tải bộ nhớ khi số lượng bước đi quá lớn, thì sử dụng phương pháp Adjoint, ta có

$$a = \frac{\partial L}{\partial S}$$

⇨ L là giá trị Loss

⇨ a là Vector ngang D chiều, gọi là Adjoint

⇨ Đầu tiên tính

$$a_N = \frac{\partial L}{\partial S_N}$$

⇨ Tiếp theo, ta có phương trình vi phân bậc 1

$$\frac{da}{dt} = -\mathrm{a}\frac{\partial f(S,\theta,t)}{\partial S}$$

⇨ Dựa vào phương trình này với trạng thái xuất phát là a_N để tính a_0, sau khi tính được a_0 là Gradient của S_0 thì dùng nó để tiếp tục lan truyền ngược về phần mạng phía sau, cách tính a_0 là thay S hiện tại, t hiện tại, a hiện tại vào phương trình vi phân, dùng đạo hàm tính được để cập nhật giá trị của a, giá trị này là giá trị trước a hiện tại 1 khoảng thời gian h, cứ như vậy cho đến thời điểm t_0 là sẽ ra a_0

⇨ Chứng minh phương trình vi phân trên, ta có

$$a_t = \frac{\partial L}{\partial S_t} = \frac{\partial L}{\partial S_{t+\varepsilon}}\frac{\partial S_{t+\varepsilon}}{\partial S_t} = a_{t+\varepsilon}\frac{\partial S_{t+\varepsilon}}{\partial S_t}$$

$$\frac{da_t}{dt} = \lim_{\varepsilon \to 0^+}\frac{a_{t+\varepsilon} - a_t}{\varepsilon} = \lim_{\varepsilon \to 0^+}\frac{a_{t+\varepsilon} - a_{t+\varepsilon}\frac{\partial S_{t+\varepsilon}}{\partial S_t}}{\varepsilon}$$

⇨ Biểu diễn $S_{t+\varepsilon}$ = chuỗi Taylor với 2 số hạng, hay ước lệ tuyến tính

$$\lim_{\varepsilon \to 0^+}\frac{a_{t+\varepsilon} - a_{t+\varepsilon}\frac{\partial S_{t+\varepsilon}}{\partial S_t}}{\varepsilon} \approx \lim_{\varepsilon \to 0^+}\frac{a_{t+\varepsilon} - a_{t+\varepsilon}\frac{\partial\left(S_t + \varepsilon f(S,\theta,t)\right)}{\partial S_t}}{\varepsilon} =$$

$$\lim_{\varepsilon \to 0^+}\frac{a_{t+\varepsilon} - a_{t+\varepsilon}\left(I + \varepsilon\frac{\partial f(S,\theta,t)}{\partial S_t}\right)}{\varepsilon} = \lim_{\varepsilon \to 0^+} -a_{t+\varepsilon}\frac{\partial f(S,\theta,t)}{\partial S_t} = -a_t\frac{\partial f(S,\theta,t)}{\partial S_t}$$

⇨ I là ma trận Identity

⇨ Bây giờ tính Gradient của θ, ta có

$$\frac{\partial L}{\partial \theta} = \frac{\partial L}{\partial S_N}\frac{\partial S_N}{\partial \theta} + \frac{\partial L}{\partial S_{N-\varepsilon}}\frac{\partial S_{N-\varepsilon}}{\partial \theta} + \frac{\partial L}{\partial S_{N-2\varepsilon}}\frac{\partial S_{N-2\varepsilon}}{\partial \theta} + \cdots + \frac{\partial L}{\partial S_\varepsilon}\frac{\partial S_\varepsilon}{\partial \theta} =$$

$$\mathrm{a}_N\frac{\partial S_N}{\partial \theta} + \mathrm{a}_{N-\varepsilon}\frac{\partial S_{N-\varepsilon}}{\partial \theta} + \cdots + \mathrm{a}_\varepsilon\frac{\partial S_\varepsilon}{\partial \theta} = \int_{t_0}^{t_N} a\frac{\partial S}{\partial \theta} = \int_{t_0}^{t_N} a\frac{f(S,\theta,t)}{\partial \theta}dt =$$

$$-\int_{t_N}^{t_0} a\frac{f(S,\theta,t)}{\partial\theta}dt$$

⇨ Từ công thức trên, dễ thấy ta có thể dùng phương pháp Euler để tìm Gradient của θ, với trạng thái xuất phát là 0, ta có thể làm điều này cùng lúc với việc tính Adjoint của S hiện tại luôn

⇨ Cuối cùng là Gradient của t_N và t_0, ta có

$$\frac{\partial L}{\partial t_N} = \frac{\partial L}{\partial S_N}\frac{\partial S_N}{\partial t_N} = \frac{\partial L}{\partial S_N}\frac{\partial S_N}{\partial t} = a_N f(S_N,\theta,t_N)$$

$$\frac{\partial L}{\partial t_0} = \frac{\partial L}{\partial S_N}\frac{\partial S_N}{\partial t_0} + \frac{\partial L}{\partial S_{N-\varepsilon}}\frac{\partial S_{N-\varepsilon}}{\partial t_0} + \frac{\partial L}{\partial S_{N-2\varepsilon}}\frac{\partial S_{N-2\varepsilon}}{\partial t_0} + \cdots + \frac{\partial L}{\partial S_\varepsilon}\frac{\partial S_\varepsilon}{\partial t_0} =$$

$$a_N\frac{\partial S_N}{\partial t} + a_{N-\varepsilon}\frac{\partial S_{N-\varepsilon}}{\partial t} + \cdots + a_\varepsilon\frac{\partial S_\varepsilon}{\partial t} = a_N f(S_N,\theta,t_N) + \int_{t_0}^{t_N} a\frac{\partial S}{\partial t} =$$

$$a_N f(S_N,\theta,t_N) - \int_{t_N}^{t_0} a\frac{f(S,\theta,t)}{\partial t}dt$$

⇨ Tính Gradient tương tự như θ

Clustering – Phân Cụm:

1. K Means Clustering (Phân Cụm K Means)?

⇨ Cho tập hợp S gồm 1 số điểm trong không gian, và số nguyên dương K cho trước, ta muốn phân chia S thành K cụm

⇨ Bước 1, chọn K điểm ngẫu nhiên trong S, vẽ sơ đồ Voronoi với K điểm này làm các điểm trung tâm

⇨ Bước 2, mỗi điểm trung tâm sẽ di chuyển tới vị trí mới, vị trí mới là trung bình tọa độ tất cả các điểm của S nằm trong tế bào Voronoi tương ứng, vẽ lại sơ đồ Voronoi, lặp đi lặp lại bước này cho tới khi không còn điểm nào trong S di cư sang tế bào Voronoi khác

⇨ Sơ đồ Voronoi cuối cùng chính là cách phân cụm hợp lí nhất cục bộ, để tìm cách phân cụm hợp lí nhất toàn cục, thì phải chạy thuật toán này nhiều lần, với K điểm trung tâm ban đầu ở vị trí khác nhau, sau đó chọn sơ đồ nào có WCSS nhỏ nhất

⇨ WCSS (Within Cluster Sum Of Squares) được tính như sau

⇨ Bước 1, gán cho mỗi điểm trong S 1 giá trị V = bình phương khoảng cách từ nó tới điểm trung tâm tương ứng

⇨ Bước 2, WCSS = tổng giá trị V của tất cả các điểm trong S

2. Elbow Method (Phương Pháp Cùi Chỏ)?

⇨ Để xác định số K tối ưu nhất trong K Means Clustering, ta vẽ đồ thị WCSS tối ưu nhất của từng K, đồ thị sẽ có dạng đi xuống khi K tăng dần, giống chữ L

⇨ K tối ưu nhất sẽ do ta chọn dựa theo cảm tính, là điểm mà ta thấy giống cái đầu gối tay hay cùi chỏ nhất

Graph – Đồ Thị:

1. Graph?

⇨ Bao gồm các đỉnh và cạnh, đỉnh là Node, cạnh có thể là kết nối 1 chiều hay 2 chiều giữa 2 Node, giữa 2 Node có thể có nhiều cạnh

⇨ Nếu tất cả các cạnh là kết nối 1 chiều thì gọi là Directed Graph

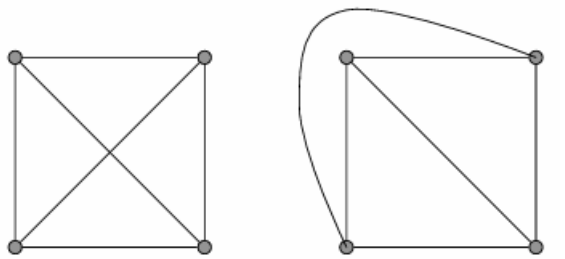
- ⇨ Directed Graph mà trong đó, không có SCC nào chứa > 1 Node, gọi là DAG (Directed Acyclic Graph)
- ⇨ Nếu tất cả các cạnh là kết nối 2 thì gọi là Undirected Graph
- ⇨ Nếu tất cả các cạnh được gán với 1 trọng số thì gọi là Weighted Graph
- ⇨ Undirected Graph mà không có cặp Node nào mà giữa chúng có nhiều hơn 1 cạnh gọi là Simple Graph, còn không thì gọi là Multigraph, bội (Multiplicity) của 1 cặp Node trong Multigraph là số cạnh giữa chúng
- ⇨ Simple Graph mà giữa 2 Node bất kì đều có 1 cạnh gọi là Complete Graph, kí hiệu K_n, n là số Node
- ⇨ Cho 1 Simple Graph G, gọi tập hợp các Node của nó là V, khi này nếu V có thể chia phần thành 2 tập con A và B, A giao B = rỗng, A hợp B = V, sao cho tất cả các cạnh trong G đều có 1 đỉnh thuộc A và 1 đỉnh thuộc B, thì G sẽ được gọi là đồ thị phân đôi (Bipartite Graph)
- ⇨ Cho 1 Simple Graph G, gọi tập hợp các Node của nó là V, khi này nếu V có thể chia phần thành 2 tập con A và B, A giao B = rỗng, A hợp B = V, sao cho ứng với mỗi cặp Node bất kì, 1 Node của A, 1 Node của B, đều có 1 cạnh giữa chúng, đồng thời không có cặp Node nào trong A có cạnh ở giữa, tương tự không có cặp Node nào trong B có cạnh giữa, khi này G sẽ được gọi là Complete Bipartite Graph, kí hiệu $K_{m,n}$, m là số Node của A và n là số Node của B
- ⇨ Multigraph mà có ít nhất 1 cạnh nối từ 1 Node tới chính nó gọi là Pseudograph, cạnh nối từ 1 Node tới chính nó gọi là khuyên (Loop)
- ⇨ 1 đường đi có thể đi lại cạnh đã đi qua, nếu không đi lại, thì được gọi là Simple Path, số cạnh 1 con đường đi qua, tính cả nếu đi lại cạnh đã đi qua, gọi là chiều dài của đường đi
- ⇨ 1 chu trình (Circuit) trong 1 Graph là 1 đường đi có điểm xuất phát và kết thúc tại cùng 1 Node
- ⇨ 1 đường đi mà đi qua tất cả các cạnh trong Graph đúng 1 lần gọi là Eulerian Path, không nhất thiết phải đi qua tất cả các Node
- ⇨ 1 chu trình mà đi qua tất cả các cạnh trong Graph đúng 1 lần gọi là Eulerian Circuit, không nhất thiết phải đi qua tất cả các Node
- ⇨ 1 đường đi mà đi qua tất cả các Node trong Graph đúng 1 lần gọi là Hamiltonian Path, không nhất thiết phải đi qua tất cả cạnh
- ⇨ 1 chu trình mà đi qua tất cả các Node trong Graph đúng 1 lần trừ điểm bắt đầu và kết thúc gọi là Hamiltonian Circuit, không nhất thiết phải đi qua tất cả cạnh
- ⇨ Chỉ có thể tìm Hamiltonian Path và Hamiltonian Circuit thông qua việc liệt kê toàn bộ Path có thể, có thể sử dụng quy hoạch động, giả sử bạn chọn Node bắt đầu và kết thúc là A, thì Node ngay trước khi kết thúc có thể là gì, liệt kê hết ra, rồi từ Node kế đó ta xác định Node trước đó nữa, …, sử dụng cách này trên Connected Simple Graph sẽ tốn thời gian là $O(n2^n)$, n là số Node
- ⇨ Undirected Graph có từ 3 Node trở lên, mà các cạnh tạo thành 1 vòng khép kín đi qua tất cả các Node, đường đi không cắt nhau, không đi lại đường đã đi, gọi là đồ thị vòng (Cycle), kí hiệu C_n, n là số Node
- ⇨ Thêm 1 Node vào chính giữa đồ thị vòng C_n, rồi nối Node này với tất cả các Node còn lại, ta được đồ thị bánh xe (Wheel), kí hiệu W_n
- ⇨ Cho 2^n Node, n từ 1 trở lên, đánh số thứ tự cho các Node bằng số nhị phân n Bit, ví dụ n = 3 thì đánh 000, 001, 010, …, 111 cho 8 Node, sau đó bất kì cặp Node nào có số thứ tự nhị phân tương ứng khác nhau đúng 1 Bit thì nối với nhau bằng

1 cạnh 2 chiều, ta được 1 Undirected Graph với tên gọi là khối n chiều (N Dimensional Hypercube), kí hiệu Q_n

⇨ Đặt 1 khối n phương đơn vị vào khít góc trục tọa độ n chiều, khi này số thứ tự của mỗi đỉnh là tọa độ của nó, thì khối này sẽ tạo thành Q_n, ví dụ n = 3, ta có khối lập phương đơn vị với tọa độ các đỉnh là (0, 0, 0), (0, 0, 1), …, (1, 1, 1)

⇨ Nếu 1 Graph có thể được vẽ vòng vèo làm sao đó sao cho không có 2 cạnh nào cắt nhau thì gọi là Planar Graph, nếu không thì gọi là Nonplanar Graph, bản vẽ của Graph sao cho không có 2 cạnh nào cắt nhau gọi là biểu diễn phẳng của Graph đó

⇨ Minh họa



⇨ Bên trái là bản vẽ thường, bên phải là hình biểu diễn phẳng của Complete Graph K_4

⇨ 1 Graph được gọi là liên thông (Connected Graph) nếu tồn tại đường đi từ 1 Node bất kì đến 1 Node bất kì

⇨ Nếu 1 Simple Graph vừa là Connected Graph vừa là Planar Graph với số Node từ 3 trở lên, thì số cạnh của nó không vượt quá 3 * số Node – 6

⇨ Nếu 1 Simple Graph vừa là Connected Graph vừa là Planar Graph với số Node từ 3 trở lên, và không có chu trình nào có chiều dài đúng = 3, thì số cạnh của nó không vượt quá 2 * số Node – 4

⇨ 2 Node a và b trong Undirected Graph mà giữa chúng có ít nhất 1 cạnh thì 2 Node này gọi là 2 Node liền kề, hay hàng xóm của nhau, kí hiệu cạnh là {a, b} hoặc {b, a}, tập hợp các hàng xóm của 1 Node a kí hiệu là N(a)

⇨ Trong Directed Graph, có 1 cạnh chĩa từ Node a tới Node b, ta nói a nối tới (Adjacent To) b, và b được nối từ (Adjacent From) a, kí hiệu cạnh là (a, b), a gọi là Node đầu (Initial), b gọi là Node cuối (Terminal)

⇨ Loop trong Directed Graph là 1 cạnh chĩa từ 1 Node tới chính nó, có Node đầu và cuối trùng nhau

⇨ Bậc của 1 Node a trong Undirected Graph là số cạnh có 1 đỉnh là Node này, riêng Loop tính là 2 cạnh, kí hiệu là deg(a), Node cô lập là Node có bậc = 0, Node treo (Pendant) là Node có bậc = 1

⇨ Bậc vào (In Degree) của 1 Node a trong Directed Graph là số cạnh chĩa vào Node này, kí hiệu $deg^-(a)$, còn bậc ra (Out Degree) của Node này là số cạnh chĩa ra từ Node này, kí hiệu $deg^+(a)$, riêng Loop tính là 1 bậc vào và 1 bậc ra

⇨ Cho 1 Undirected Graph bất kì, ta có các định lý sau

Tổng số bậc của tất cả các Node = 2 lần tổng số cạnh Tổng số Node có bậc lẻ là 1 số chẵn Nếu số Node là lẻ thì số Node có bậc chẵn sẽ là 1 số lẻ Nếu số Node là chẵn thì số Node có bậc chẵn sẽ là 1 số chẵn

⇨ Cho 1 Directed Graph bất kì, ta có các định lý sau

Tổng số bậc vào của tất cả các Node = tổng số bậc ra của tất cả các Node = tổng số cạnh

⇨ Cho 1 Connected Multigraph bất kì, ta có các định lý sau

Tồn tại Eulerian Path khi và chỉ khi tất cả các Node bậc chẵn hoặc có đúng 2 Node bậc lẻ
Tồn tại Eulerian Circuit khi và chỉ khi tất cả các Node bậc chẵn
Nếu có nhiều hơn 2 Node bậc lẻ, thì không tồn tại Eulerian Path

⇨ Cho 1 Directed Graph bất kì sao cho khi thay thế tất cả các cạnh có hướng thành vô hướng, ta được 1 Connected Graph

Tồn tại Eulerian Circuit khi và chỉ khi tất cả các Node có bậc vào = bậc ra
Tồn tại Eulerian Path khi và chỉ khi số lượng Node có bậc ra – bậc vào = 1 không vượt quá 1, số lượng Node có bậc vào – bậc ra = 1 không vượt quá 1, và tất cả các Node còn lại có bậc vào = bậc ra

⇨ Cho 1 Complete Graph K_n, ta có các định lý sau

Số cạnh = số Node * (số Node – 1) / 2
Số Hamilton Circuit khác nhau = (n – 1)! / 2

⇨ Để biểu diễn 1 Graph G, ta dùng danh sách kề (Adjacency List), giống như Dictionary, mỗi Key là 1 Node, Value là những Node có thể đi tới trực tiếp từ Node này, kí hiệu là G = (V, E), V là tập hợp Node, E là tập hợp các cạnh

⇨ Ta cũng có thể biểu diễn Graph G = (V, E) bằng ma trận kề (Adjacency Matrix) kích thước số Node x số Node, kí hiệu A_G, hàng m cột n sẽ = 1 nếu có cạnh chĩa ra từ Node thứ m vào Node thứ n, còn không thì = 0

⇨ Ta cũng có thể biểu diễn Graph G = (V, E) bằng ma trận liên thuộc (Incidence Matrix) kích thước số Node x số cạnh, kí hiệu M_G, hàng m cột n sẽ = 1 nếu Node thứ m là 1 đỉnh của cạnh thứ n, còn không thì = 0

2. Graph Con (Subgraph)?

⇨ Cho Graph G = (V, E), khi này G' = (V', E') được gọi là Graph con của G nếu V' là tập con của V và E' là tập con của E

⇨ 1 Simple Graph là Nonplanar Simple Graph khi và chỉ khi nó chứa Graph con đồng phôi với Complete Bipartite Graph $K_{3,3}$ hoặc Complete Graph K_5

3. Hợp Graph?

⇨ Cho Grapg G = (V, E) và Graph G' = (V', E'), khi này hợp giữa 2 Graph này là $G \cup G' = (V \cup V', E \cup E')$

4. Phân Chia Sơ Cấp (Elementary Subdivision)?

⇨ Cho 1 Planar Undirected Graph G, phân chia sơ cấp trên G là hành động chấm 1 điểm vào chính giữa 1 cạnh và đặt 1 Node mới vào điểm đó, như vậy từ 1 cạnh ban đầu phân ra 2 cạnh nhỏ

⇨ 2 Planar Undirected Graph G và F được gọi là đồng phôi (Homeomorphic) nếu tồn tại một Planar Undirected Graph H sao cho từ H, thực hiện 1 số phân chia sơ cấp ta sẽ có được G, và cũng từ H, thực hiện 1 số phân chia sơ cấp khác ta sẽ có được F

5. Graph Isomorphism?

⇨ Cho f là 1 ánh xạ từ Graph G tới Graph H, nếu bản chất là H được tạo ra bằng cách thay đổi vị trí vẽ các Node trong G, thay đổi tên Node, thay đổi tên cạnh, thì f gọi là phép đồng cấu (Isomorphism), G và H gọi là đồng cấu (Isomorphic) với nhau

⇨ Trường hợp 2 Node, có tất cả 2 Simple Graph không đồng cấu với nhau

⇨ Trường hợp 3 Node, có tất cả 4 Simple Graph không đồng cấu với nhau
⇨ Trường hợp 4 Node, có tất cả 11 Simple Graph không đồng cấu với nhau

6. Tìm Eulerian Path Và Eulerian Circuit Bằng Thuật Toán Fleury?

⇨ Để tìm 1 Eulerian Circuit P của 1 Connected Multigraph G, đảm bảo rằng G có Eulerian Circuit
⇨ Bước 1, đứng tại 1 Node bất kì, chèn Node này vào P
⇨ Bước 2, từ Node này đi theo 1 cạnh bất kì không phải cạnh cắt của G, nếu chỉ có 1 cạnh thì đi theo luôn đéo quan tâm cắt hay không cắt, sau đó xóa cạnh này khỏi G, vẫn giữ lại Node ở 2 đầu, chèn Node hiện tại vào P, lặp lại bước này cho tới khi trở về vị trí ban đầu
⇨ Để tìm 1 Eulerian Path P của 1 Connected Multigraph G, đảm bảo rằng G có Eulerian Path, tương tự như khi tìm Eulerian Circuit, nhưng Node ban đầu phải là Node bậc lẻ, nếu không có Node bậc lẻ thì chọn Node bất kì, đồng thời điều kiện dừng là G hết cạnh

7. Tìm Eulerian Path Và Eulerian Circuit Bằng Thuật Toán Hierholzer?

⇨ Cho 1 Directed Graph G bất kì sao cho khi thay thế tất cả các cạnh có hướng thành vô hướng, ta được 1 Connected Graph
⇨ Để tìm Eulerian Path P của G, đảm bảo rằng G có Eulerian Path
⇨ Bước 1, đứng tại Node có bậc ra – bậc vào = 1, nếu không có Node này thì đứng tại Node bất kì
⇨ Bước 2, thực hiện DFS từ Node này, cứ mỗi lần quay lui khỏi 1 Node thì chèn Node đó vào bên trái P
⇨ Cuối cùng đọc các phần tử của P từ trái sang, ta được 1 Eulerian Path
⇨ Để tìm Eulerian Circuit P của G, đảm bảo rằng G có Eulerian Circuit, làm tương tự như khi tìm Eulerian Path, nhưng Node khởi đầu là Node bất kì
⇨ Có thời gian thực thi là O(E), E là số cạnh của G

8. TSP (Traveling Salesman Problem)?

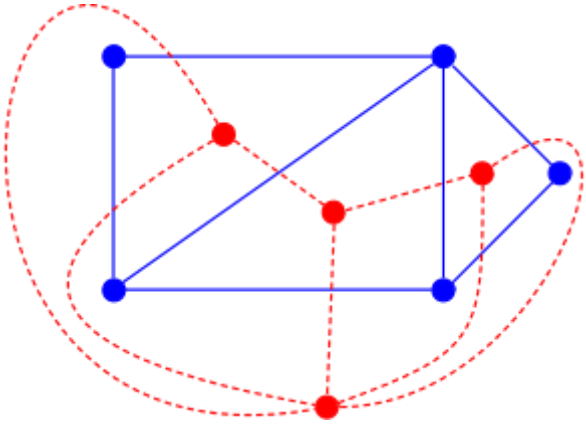
⇨ Cho 1 Connected Simple Graph, mỗi cạnh có trọng số ứng với nó, bài toán đặt ra là tìm Hamilton Circuit với tổng trọng số trên đường đi là nhỏ nhất
⇨ Sử dụng quy hoạch động, giả sử bắt đầu và kết thúc từ Node A, vậy thì Node trước đó có thể là gì, liệt kê hết ra, đồng thời lưu trọng số khi đi từ những Node này tới A, rồi cứ lan truyền ngược lên trên
⇨ Thời gian để giải khi sử dụng quy hoặc động là $O(n^2 2^n)$

9. VRP (Vehicle Routing Problem)?

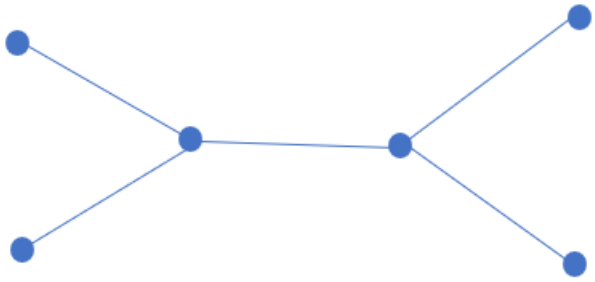
⇨ Cho 1 Weighted Simple Graph gồm các Node A_1, A_2, … là các kho hàng của bạn, các Node còn lại là vị trí cần giao hàng, trọng số của cạnh là khoảng cách giữa 2 Node tương ứng, giả sử đơn vị km, giả sử bạn có chừng K xe vận chuyển y chang nhau, mỗi xe chỉ chở được tối đa X đơn hàng và chỉ đi được tối đa Y km, giả sử kho hàng của bạn có Z đơn hàng, Z là số Node cần giao tới, mỗi Node nhận 1 đơn hàng
⇨ Bài toán đặt ra là phân phối số lượng K xe vào từng kho hàng, đồng thời tìm đường đi cho tất cả K xe, sao cho tất cả các Node đều nhận được đơn hàng của họ, với tiêu chí là tổng quãng đường đi của tất cả xe là nhỏ nhất, lưu ý điểm bắt đầu và kết thúc hành trình mỗi xe phải là kho hàng nó thuộc về
⇨ Nếu số lượng kho hàng là 1 thì bài toán gọi là Single Dopot Capacitate VRP, khi này không cần tính toán số xe cần phân phối vào mỗi kho hàng

10. Dual Graph?

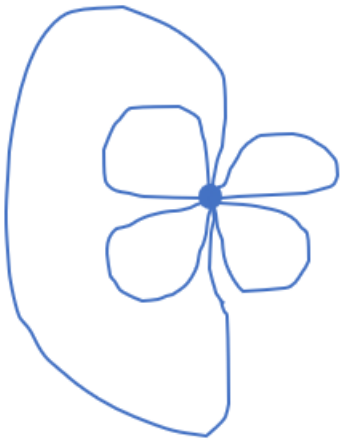
- ⇨ Cho 1 Planar Undirected Graph G, Dual Graph G’ của nó là 1 Planar Undirected Graph được tạo ra bằng cách sau
- ⇨ Với mỗi vùng kín trong G, chấm 1 điểm ở giữa, Loop cũng tính là vùng kín
- ⇨ Với mỗi cạnh trong G, giả sử cạnh này là tiếp giáp giữa vùng kín A và B, thì nối 2 điểm đã chấm tương ứng trong A và B với nhau, nếu A và B là 1, thì vẽ Loop
- ⇨ Cuối cùng ta sẽ có được G’
- ⇨ Dual Graph của G’ là G
- ⇨ Ví dụ 1



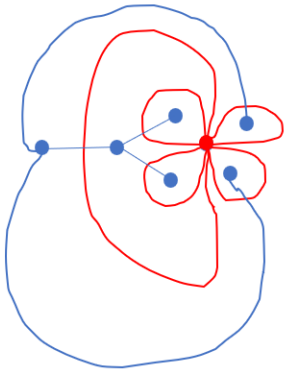
- ⇨ G là đường xanh, G’ là đường đỏ
- ⇨ Để ý thấy vùng bên ngoài cũng tính là 1 vùng kín
- ⇨ Ví dụ 2



- ⇨ Để ý thấy Graph này chỉ có 1 vùng kín duy nhất là toàn bộ mặt phẳng, mỗi cạnh ta sẽ vẽ 1 Loop



- ⇨ Từ Dual Graph này, ta lấy Dual Graph của nó, sẽ được Graph ban đầu



11. Tô Màu Graph?
⇨ 1 tô màu của 1 Undirected Graph là 1 bản vẽ của Graph đó, với các Node được tô màu sao cho không có 2 Node hàng xóm nào cùng màu
⇨ Số màu ít nhất dùng để tạo ra 1 tô màu của 1 Undirected Graph G gọi là số màu (Chromatic Number) của G, kí hiệu $\chi(G)$
⇨ Số màu của 1 Planar Undirected Graph luôn từ 4 trở xuống
12. Complement Graph?
⇨ Cho Complete Graph K_n, từ nó phân ra 2 Graph con, sao cho cả 2 đều có Full Node của K_n, hợp Graph của 2 Graph con này = K_n và 2 Graph con này không có cạnh chung nào, thì chúng được gọi là Complement Graph của nhau
⇨ Kí hiệu là G^c là Complement Graph của G
13. Union Find?
⇨ Là 1 Graph gồm nhiều Undirected Graph rời rạc
⇨ Để hợp 2 Undirected Graph lại, thì chỉ cần biến 1 Node bất kì trong Undirected Graph này thành con của 1 Node khác trong Undirected Graph kia
14. Cây Khung (Spanning Tree)?
⇨ Cho 1 Connected Undirected Graph G bất kì, Graph con nào của G mà nó chứa toàn bộ Node của G, đồng thời là 1 Tree thì gọi là 1 Spanning Tree của G
⇨ Có thể dễ dàng tìm 1 Spanning Tree bằng DFS hoặc BFS
⇨ Cho 1 Connected Undirected Weighted Graph G bất kì, Spanning Tree nào của G mà có tổng trọng số trên tất cả các cạnh của nó là nhỏ nhất thì gọi là Minimum Spanning Tree
15. Tìm Mimimum Spanning Tree = Thuật Toán Prim?
⇨ Cho 1 Connected Undirected Weighted Graph G, để tìm Miminum Spanning Tree của nó, thực hiện các bước sau
⇨ Bước 1, chọn 1 Node A bất kì làm điểm xuất phát, tô màu đỏ cho A
⇨ Bước 2, chọn cạnh đi từ A với trọng số nhỏ nhất, nếu có nhiều trọng số cùng nhỏ nhất thì chọn cạnh bất kì trong số chúng, đi theo cạnh này, tô màu đỏ cho cạnh này và Node đích
⇨ Bước 3, trong tất cả các cạnh đi ra từ 1 Node trong số các Node màu đỏ tới 1 Node khác không có màu đỏ, chọn cạnh có trọng số nhỏ nhất, nếu có nhiều trọng số cùng nhỏ nhất thì chọn cạnh bất kì trong số chúng, đi theo cạnh này, tô màu đỏ cho cạnh này và Node đích, lặp lại bước này cho tới khi tất cả các Node đều màu đỏ
⇨ Khi này tất cả các cạnh màu đỏ tạo thành Minimum Spanning Tree của G
16. Tìm Mimimum Spanning Tree = Thuật Toán Kruskal?

⇨ Cho 1 Undirected Weighted Graph G, G có thể có nhiều Connected Component, khi này ta muốn tìm Miminum Spanning Tree của tất cả Component, thực hiện các bước sau
⇨ Bước 1, chọn cạnh có trọng số nhỏ nhất, nếu có nhiều trọng số cùng nhỏ nhất thì chọn cạnh bất kì trong số chúng, tô màu đỏ cho cạnh này và 2 Node nó nối
⇨ Bước 2, tiếp tục chọn cạnh có trọng số nhỏ nhất, chỉ chọn trong những cạnh mà 2 Node ở đầu không đồng thời có màu đỏ, nếu có nhiều trọng số cùng nhỏ nhất thì chọn cạnh bất kì trong số chúng, tô màu đỏ cho cạnh này và 2 Node nó nối, lặp lại bước này cho tới khi toàn bộ Node đều màu đỏ
⇨ Khi này các cạnh màu đỏ sẽ tạo thành các Minimum Spanning Tree của các Connected Component trong G

Data Structure – Cấu Trúc Dữ Liệu:

1. Tree?
⇨ Là Connected Simple Graph không chứa Simple Circuit
⇨ Có thể biểu diễn phân nhánh từ trên xuống
2. Forest?
⇨ Là Simple Graph không chứa Simple Circuit
⇨ Bản chất là 1 đống Tree
⇨ Cho 1 Forest bất kì gồm n Tree, ta có các định lý sau

Số cạnh = số Node – n

3. Rooted Tree?
⇨ Là 1 Tree nhưng thay tất cả các cạnh vô hướng thành cạnh có hướng, tức là nó là Directed Graph, hướng đi sẽ xuất phát từ Node được chọn làm gốc và cứ thế lan truyền
⇨ Cho 1 Rooted Tree, ta có các định nghĩa sau
⇨ Node A được gọi gọi là cha (Parent) của Node B nếu có cạnh chĩa từ A đến B, khi này B gọi là con (Child) của A
⇨ 2 Node có chung cha gọi là anh em (Sibling) của nhau
⇨ Node anh em của Node cha thì gọi là Node bác (Uncle)
⇨ Node cha của Node cha thì gọi là Node ông nội (Grandparent)
⇨ Node A được gọi là tổ tiên (Ancestor) của Node B khác A, nếu từ A có thể đi được đến B, khi này B được gọi là con cháu (Descendant) của A
⇨ Leaf Node là Node không có con
⇨ Đỉnh trong (Internal Vertex) là Node có ít nhất 1 con
⇨ Mức (Level) của 1 Node A là số cạnh của đường đi từ Node gốc tới A
⇨ Độ cao (Height) của Rooted Tree là mức lớn nhất
4. Cấy (Transplant)?
⇨ Cho 1 Rooted Tree, và 2 Node A và B trong nó, A không phải là con cháu của B, khi này hành động cấy B vào A nghĩa là cắt nhánh Tree con với gốc B và thế nó vào vị trí của A, nhánh Tree con với gốc A bị loại bỏ
5. Ordered Rooted Tree?
⇨ Là 1 Rooted Tree kèm theo 1 quy ước thứ tự
6. M Ary Tree?
⇨ Là Rooted Tree mà tất cả các Node của nó có không quá M Node con

⇨ Nếu M = 2 thì gọi là cây nhị phân (Binary Tree)
⇨ Số Leaf Node tối đa mà 1 M Ary Tree với độ cao h có thể có là M^h

7. Balanced M Ary Tree?

⇨ Là M Ary Tree G mà mức của tất cả Leaf Node ≥ độ cao của G – 1
⇨ Trong 1 M Ary Tree, 1 Node được gọi là cân bằng nếu Tree con với gốc là nó là 1 Balanced M Ary Tree

8. Full M Ary Tree?

⇨ Là Rooted Tree mà tất cả các đỉnh trong của nó đúng M Node con
⇨ Cho 1 Full M Ary Tree bất kì, ta có các định lý sau

Số đỉnh trong = (số Node – 1) / M Số Leaf Node = (số Node * (M – 1) + 1) / M

⇨ Cho 1 Balanced Full M Ary Tree, ta có công thức sau, h là độ cao của nó, n là số Leaf Node của nó

$h = \lceil log_m(n) \rceil$

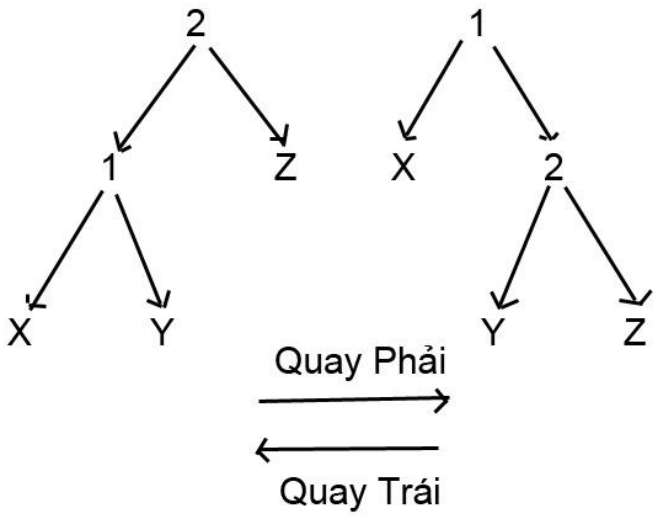
9. Complete M Ary Tree (Perfect M Ary Tree)?

⇨ Là Full M Ary Tree mà tất cả Leaf Node có cùng mức

10. Ordered Binary Tree?

⇨ Là Binary Tree kèm theo quy ước thứ tự như sau, Node con thứ nhất gọi là con bên trái (Left Child), Node con thứ 2 gọi là con bên phải (Right Child)

11. Quay 1 Ordered Binary Tree?



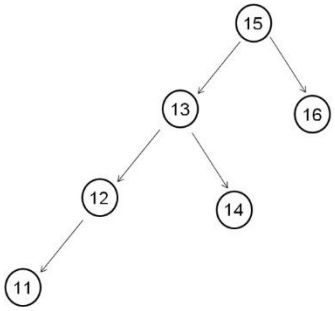
⇨ X, Y, Z là những Tree con bất kì, có thể là Null
⇨ Để ý thấy giống như ta đang trượt X, 1, 2, Z
⇨ Cho Node X, Node Y là con của X, ta có các phép quay sau

Zig(X) = phép quay phải 1 lần với tâm quay X Zag(X) = Zig(X) nhưng quay trái Zig Zig(X) = phép quay phải 1 lần với tâm quay X, sau khi Y thế chỗ X thì quay phải lần nữa với tâm quay Y Zag Zag(X) = Zig Zig(X) nhưng Full quay trái Zig Zag(X) = phép quay phải với tâm quay Y, sau đó quay trái với tâm quay X Zag Zig(X) = phép quay trái với tâm quay Y, sau đó quay phải với tâm quay X

12. BST (Binary Search Tree, Cây Tìm Kiếm Nhị Phân)?

⇨ Là Binary Tree mà với mọi Node A, giá trị của mọi Node thuộc Tree con bên trái của A < giá trị của A, và giá trị của mọi Node thuộc Tree con bên phải của A > giá trị của A

⇨ Ví dụ



⇨ Thời gian tìm kiếm trường hợp ngon nhất là O(log(N)), còn tồi tệ nhất là O(N), N là số Node

⇨ Để xóa 1 Node X ra khỏi BST

⇨ Bước 1, tìm vị trí trí của Node X

⇨ Bước 2, ta có 3 trường hợp sau

⇨ Trường hợp 1, nếu X là Leaf Node, xóa X

⇨ Trường hợp 2, nếu X chỉ có 1 Node con, cấy Node con này vào X

⇨ Trường hợp 3, nếu X có 2 Node con, tìm Node có giá trị nhỏ nhất trong Tree con bên phải của X hoặc Node có giá trị lớn nhất trong Tree con bên trái của X, gọi là Y, gán giá trị của Y cho X, rồi xóa Y

13. Cây Đỏ Đen (Red Black Tree)?

⇨ Là 1 BST A kèm thuộc tính màu cho mỗi Node, tức là mỗi Node phải có màu đỏ hoặc đen, và nó phải thỏa mãn các điều kiện sau, gọi B là BST được tạo ra từ A bằng cách thêm các Null Node vào A để Node nào trong A cũng có 2 Node con, các Null Node quy ước có màu đen

Node gốc phải có màu đen Mỗi Node đỏ trong B phải có các Node con toàn đen Trên đường đi từ 1 Node C bất kì trong B, đến mọi Null Node là con cháu của C, phải có số Node đen bằng nhau

⇨ Thời gian tìm kiếm trong cây đỏ đen luôn là O(log(N)), N là số Node

⇨ Để chèn 1 Node X vào cây đỏ đen

⇨ Bước 1, chèn Node X vào cây đỏ đen như khi chèn vào BST, gán cho nó màu đỏ

⇨ Bước 2, nếu Node cha hiện tại của X, gọi là P, có màu đỏ, thì xét các trường hợp sau, gọi G và U lần lượt là Node ông nội và Node bác hiện tại của X

⇨ Trường hợp 1, nếu U màu đỏ thì gán lại màu đen cho U và P, còn G thì gán lại màu đỏ, đồng thời kể từ bây giờ Node G được xem là Node X

⇨ Trường hợp 2, nếu U màu đen, thì phân ra 4 trường hợp con sau

⇨ Trường hợp 2.1, nếu P bên trái G và X bên phải P thì làm phép Zag(P), và kể từ bây giờ Node P được xem là Node X

⇨ Trường hợp 2.2, nếu P bên phải G và X bên trái P thì làm phép Zig(P), và kể từ bây giờ Node P được xem là Node X

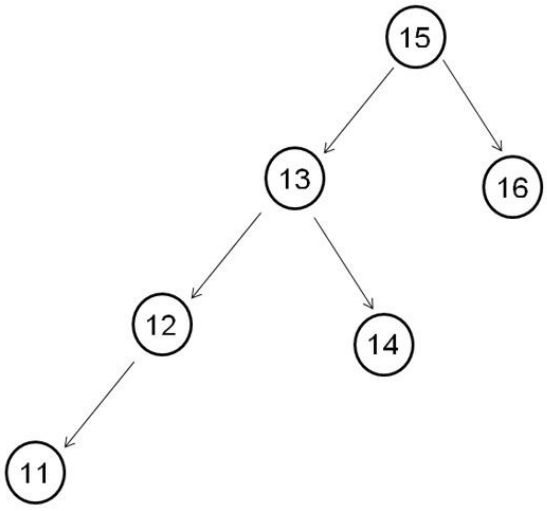
⇨ Trường hợp 2.3, nếu P bên trái G và X bên trái P thì làm phép Zig(G), đồng thời đổi màu của G sang đỏ và P sang đen

⇨ Trường hợp 2.4, nếu P bên phải G và X bên phải P thì làm phép Zag(G), đồng thời đổi màu của G sang đỏ và P sang đen

⇨ Bước 3, lặp lại bước 2 cho đến khi nào Node cha của X không còn màu đỏ
⇨ Bước 4, gán lại màu đen cho Node gốc
⇨ Để xóa 1 Node X ra khỏi cây đỏ đen A
⇨ Bước 1, thêm các Null Node vào A để cho Node nào cũng có 2 Node con
⇨ Bước 2, tìm Node X trong A, có 3 trường hợp
⇨ Trường hợp 1, 2 Node con của X đều là Null Node, đặt Y là Null Node con bên phải X, cấy Y vào X, Y đồng thời cũng là Node thay thế
⇨ Trường hợp 2, X có đúng 1 Node con là Null Node, cấy Node con còn lại vào X, đặt Y là Node con này, Y đồng thời cũng là Node thay thế
⇨ Trường hợp 3, X có đủ 2 Node con không phải Null Node, tìm Node không phải Null Node có giá trị nhỏ nhất ở Tree con bên phải của X, đây là Node thay thế, đặt Y là Node con bên phải của Node thay thế, thế X thành giá trị của Node thay thế, đồng thời cấy Y vào Node thay thế
⇨ Bước 3, xét các trường hợp sau
⇨ Trường hợp 1, X màu đỏ, Node thay thế màu đỏ hoặc là Null Node, dừng thuật toán
⇨ Trường hợp 2, X màu đen, Node thay thế màu đỏ, sau khi thay thế thì tô Node thay thế thành màu đen, dừng thuật toán
⇨ Trường hợp 3, X màu đỏ, Node thay thế màu đen, sau khi thay thế thì tô Node thay thế thành màu đỏ, tiếp tục bước 4
⇨ Trường hợp còn lại không làm gì cả, tiếp tục bước 4
⇨ Bước 4, xét các trường hợp sau
⇨ Trường hợp 1, Y màu đỏ, tô Y thành màu đen, dừng thuật toán
⇨ Trường hợp 2, Y màu đen hoặc là Null Node, Node anh em của Y là W màu đỏ, tô W thành đen, tô Node cha của Y là P thành đỏ, nếu Y bên trái P, làm phép Zag(P), nếu Y bên phải P, làm phép Zig(P)
⇨ Trường hợp 3, Y màu đen hoặc là Null Node, Node anh em của Y là W màu đen hoặc là Null Node, chia ra 3 trường hợp con sau
⇨ Trường hợp con 3.1, 2 Node con của W đều đen hoặc là Null Node, tô W đỏ, nếu Node cha của Y là P màu đỏ, tô P đen rồi dừng thuật toán, nếu P đen thì kể từ bây giờ P được xem là Y
⇨ Trường hợp con 3.2, W có đúng 1 Node con màu đỏ và 1 Node con màu đen hoặc là Null Node, sao cho nếu Y là Node con bên trái của P, P là Node cha của nó, thì Node con màu đỏ của W nằm ở bên trái nó, và nếu Y là Node con bên phải của P, thì Node con màu đỏ của W nằm ở bên phải nó, khi này ta sẽ tô Node con màu đỏ của W thành đen, tô W thành đỏ, nếu Y bên trái P thì làm phép Zig(W), nếu Y bên phải P thì làm phép Zag(W)
⇨ Trường hợp con còn lại khi các trường hợp con trên không thỏa mãn, tô W giống màu với Node cha của Y là P, tô P thành đen, nếu Y bên trái P tô Node con bên phải của W thành đen, rồi làm phép Zag(P), nếu Y bên phải P tô Node con bên trái của W thành đen, rồi làm phép Zig(P), dừng thuật toán
⇨ Bước 5, lặp lại bước 4 cho tới khi Y trở thành Node gốc

14. AVL Tree (Cây Tìm Kiếm Nhị Phân Tự Cân Bằng)?

⇨ Là 1 BST mà tất cả các Node của nó đều cân bằng
⇨ Ví dụ
⇨ Đây không phải là AVL Tree



⇨ Thời gian tìm kiếm của AVL Tree luôn là O(log(N))
⇨ Để chèn 1 Node vào AVL Tree
⇨ Bước 1, chèn Node X vào như trong BST
⇨ Bước 2, từ X lan truyền ngược lên, gặp phải Node Y đầu tiên không cân bằng thì dừng lại, gọi đường đi từ X đến Y là C, gọi Node kế Y trên C là Z
⇨ Bước 3, nếu X thuộc nhánh con bên phải của Z và Z là Node con bên phải của Y thì thực hiện phép Zag với tâm quay Y, nếu nằm bên trái hết thì Zig, nếu X thuộc nhánh con bên phải của Z và Z là Node con bên trái của Y thì thực hiện phép Zag Zig với tâm quay là Z vầ Y, ngược lại thì Zig Zag
⇨ Làm xong 3 bước là toàn bộ Tree cân bằng

15. Cách Xóa 1 Node Trong AVL Tree?

⇨ Ban đầu, xóa như khi xóa với BST, nghĩa là sẽ thế Node bị xóa = Node nhỏ nhất lớn hơn nó, nếu không có thì Node lớn nhất nhỏ hơn nó, gọi Node này là W
⇨ Sau đó, từ Node cha của W lan truyền ngược, và quá trình cân bằng AVL Tree đéo khác gì như khi chèn Node

16. Splay 1 Node Trong BST?

⇨ Là động tác đưa 1 Node trong BST trở thành Node gốc
⇨ Gọi Node muốn đưa làm gốc là X
⇨ Bước 1, xác định Node cha và Node ông nội của X, lần lượt là Y và Z
⇨ Bước 2, nếu Y và Z không tồn tại thì dừng thuật toán
⇨ Bước 3, nếu Z không tồn tại thì thực hiện 1 phép Zig hoặc Zag để quay quanh tâm Y, sao cho X được đưa lên trên, dừng thuật toán
⇨ Bước 4, nếu X bên phải Y và Y bên phải Z thì thực hiện 1 phép Zag Zag để đưa X thế chỗ Z, nếu X bên trái Y và Y bên trái Z thì dùng phép Zig Zig, nếu X bên phải Y và Y bên trái Z thì dùng phép Zag Zig, còn không thì Zig Zag
⇨ Bước 5, quay lại bước 1

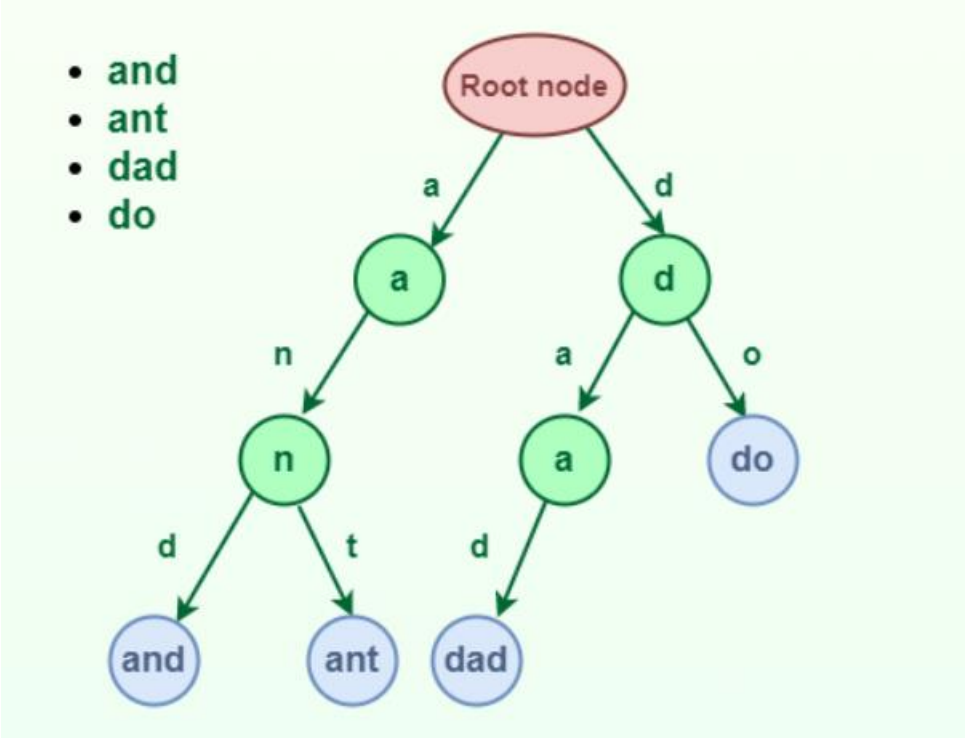
17. Splay Tree?

⇨ Là BST mà Node được thêm vào sau cùng sẽ làm Node gốc, dễ dàng truy xuất những Node mới thêm
⇨ Để chèn 1 Node vào Splay Tree
⇨ Chèn như trong BST + Splay Node đó lên làm Node gốc
⇨ Để tìm kiếm 1 Node trong Splay Tree
⇨ Tìm kiếm như trong BST + Splay Node tìm được lên làm Node gốc, nếu không tìm được thì Splay Node cuối cùng trên hành trình tìm kiếm

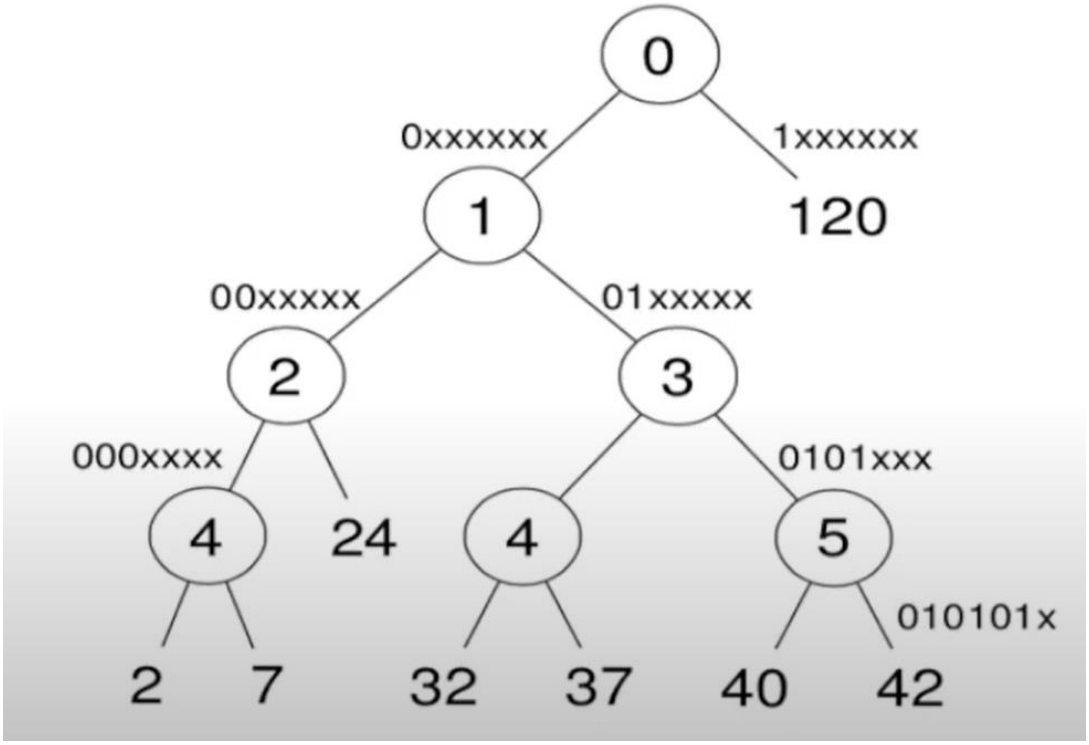
- ⇨ Để xóa 1 Node khỏi Splay Tree
- ⇨ Tìm kiếm Node đó + sau khi nó được Splay làm Node gốc thì xóa Node đó + khi này sẽ có 2 BST bị rời bên trái và bên phải + Splay Node có giá trị lớn nhất của BST bên trái, nói cách khác là Node phải cùng của BST bên trái lên làm Node gốc của BST bên trái + do nó lớn nhất rồi nên Node con bên phải không tồn tại + gán Node con bên phải của nó thành Node gốc của BST bên phải
- ⇨ Việc phải Splay Node phải cùng là để cho Splay Tree được cân bằng, do Node này có giá trị ở gần giữa
- ⇨ Khi Splay Tree đủ lớn, thời gian để thực hiện việc tìm kiếm, chèn, xóa đều = O(log(N)), nhưng nếu tìm kiếm 1 Node nhiều lần thì kể từ lần thứ 2, thời gian tìm kiếm = O(1)

18. Trie?

- ⇨ Là 1 cây có đa dạng số nhánh, giá trị của Node lá = giá trị tổng hợp được từ các Node trên đường đi từ Node gốc tới nó
- ⇨ Dữ liệu thật được lưu trữ ở Node lá, các Node trên đường đi không quan tâm
- ⇨ Ví dụ
- ⇨ Prefix Trie



- ⇨ Pat Trie (dựa vào các Bit nhị phân, được rút ngắn sao cho không có Node nào có đúng 1 Node con)



19. B Tree?

⇨ Mở rộng của BST, mỗi Node thay thành 1 Block, 1 Block có m – 1 ô, mỗi cạnh ô vuông có thể vẽ ra 1 nhánh, do đó 1 Block có tối đa m nhánh, 1 ô chứa tối đa 1 giá trị

⇨ Giá trị nhánh phải nằm giữa 2 ô nó thuộc về

⇨ Giá trị trong các ô trong 1 Block được sắp xếp theo thứ tự tăng dần

⇨ Mỗi Block không phải lá sẽ phải có ít nhất m / 2 làm tròn lên nhánh

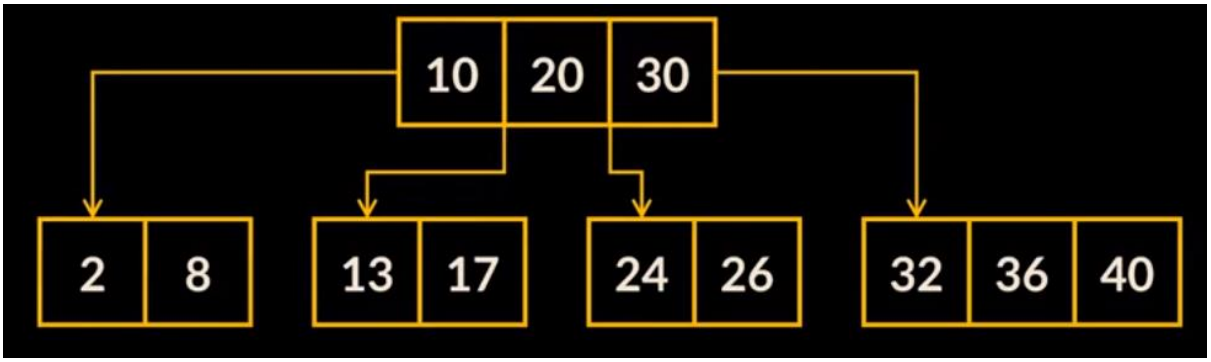
⇨ Mỗi Block không phải lá có n giá trị thì phải có đúng n + 1 nhánh

⇨ Mỗi Block không phải Block gốc phải bao gồm ít nhất m / 2 làm tròn lên – 1 giá trị

⇨ Tất cả Block lá phải ở độ cao = nhau

⇨ Ví dụ

⇨ B Tree bậc 4 (m = 4)



⇨ Để chèn thêm 1 giá trị vào B Tree

⇨ Chèn như BST nhớ sắp xếp tăng dần + sau khi chèn nếu Block bị dư 1 giá trị thì lấy giá trị của ô ở giữa Block đưa lên trên 1 bậc, dựa vào vị trí nhánh để chèn lên + nếu vẫn dư thì tiếp tục làm cho tới đỉnh

⇨ Để xóa 1 giá trị khỏi B Tree

⇨ Xóa giá trị đó khỏi ô + biến đổi kiểu gì sao cho thỏa mãn tất cả điều kiện của B Tree

20. Hash Table (Bảng Băm)?
⇨ Dùng để ánh xạ Key với Value
⇨ Ví dụ
⇨ Ánh xạ tên của 1 người với tuổi của người đó
⇨ Đầu tiên, tạo ra 1 Hash Table A với kích thước cố định N rồi đánh Index từng ô, đồng thời tạo ra 1 mảng B có kích thước tương tự, mỗi phần tử trong B sẽ là 1 Linked List
⇨ Tiếp theo, khi chèn thêm 1 cặp Key Value thì dùng 1 hàm Hash để ánh xạ Key tới 1 Index trong A, Linked List tương ứng trong B với cùng Index sẽ chèn thêm Value của Key này vào, đồng thời cả Key nữa
⇨ Mỗi khi mà ta muốn biết Value của Key K, thì chỉ cần cho Key K vào hàm Hash, được Index, truy cập Linked List có Index này trong B, rồi dò Value nào có Key trùng với K
21. Heap (Đống)?
⇨ Là Binary Tree cân bằng hoàn hảo, nghĩa là hàng nào cũng Full Node, trừ hàng cuối các Node được Fill từ trái sang phải
⇨ Các Node được sắp xếp theo 1 thứ tự nhất định, hoặc Node cha > Node con, gọi là Max Heap, hoặc Node cha < Node con, gọi là Min Heap
22. Cách Chèn Node Vào Heap?
⇨ Ở đây ta xét Max Heap, Min Heap làm tương tự
⇨ Ban đầu, chèn Node như chèn vào Binary Tree hoàn chỉnh, sau đó, so sánh Node được chèn với Node cha của nó, xem thằng nào lớn hơn, nếu Node được chèn lớn hơn, thì hoán vị nó với Node cha, tiếp tục làm như vậy cho tới khi Node được chèn trở thành gốc hoặc Node cha > Node được chèn
23. Cách Xóa Node Khỏi Heap?
⇨ Bạn chỉ được phép xóa Node gốc
⇨ Ban đầu, xóa Node gốc ra khỏi Heap, rồi đưa Leaf Node cuối cùng là A lên gốc, sau đó so sánh 2 Node con A, cái nào lớn hơn thì cho so sánh với A, nếu nó > A thì hoán vị nó với A, tiếp tục làm như vậy cho tới khi A trở lại thành Leaf Node hoặc A > 2 Node con
24. Heapify (Vun Đống)?
⇨ Là việc bạn biến 1 Binary Tree hoàn chỉnh thành 1 Heap
⇨ Ở đây ta tạo Max Heap, Min Heap tương tự
⇨ Gọi biểu diễn mảng của Binary Tree hoàn chỉnh đó là A, bắt đầu từ Node cuối cùng của A chạy ngược về Node đầu, nói cách khác từ Leaf Node cuối cùng đến Node gốc của Binary Tree, đi đến Node nào thì xác định nhánh ứng với Node đó, đưa Node này xuống dưới y chang như khi bạn xóa Node gốc khỏi Heap, sau đó nhánh này chắc chắn sẽ trở thành 1 Heap
⇨ Cuối cùng bạn sẽ thu được Max Heap
⇨ Thời gian thực thi của Heapify là O(n)
⇨ Chứng minh
⇨ Gọi tổng số Node là n
⇨ Dễ thấy, mỗi Leaf Node bản chất là 1 Heap nên ta có thể bỏ qua
⇨ Số Node ở hàng phía trên Leaf Node = n / 4, quãng đường di chuyển xuống = 1 Node
⇨ Số Node ở hàng phía trên nữa = n / 8, quãng đường di chuyển xuống = 2 Node
⇨ Số Node ở hàng phía trên nữa = n / 16, quãng đường di chuyển xuống = 3 Node
⇨ …

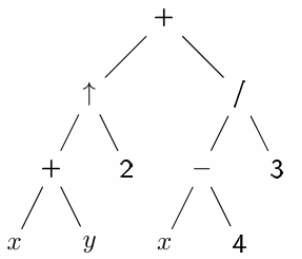
⇨ Dễ thấy, tổng quãng đường đi là

$$\frac{n}{4} + 2 * \frac{n}{8} + 3 * \frac{n}{16} + 4 * \frac{n}{32} + \cdots = \frac{1}{2} n \sum_{i=1}^{\infty} \frac{i}{2^i} = \frac{1}{2} n * 2 = n$$

⇨ Vậy thời gian thực thi là O(n)

Traversal – Duyệt Cây:

1. DFS (Depth First Search, Tìm Kiếm Theo Chiều Sâu)?
⇨ Là cách duyệt 1 Graph theo kiểu tràn nước lũ, nói cách khác là đệ quy
⇨ Có thời gian thực thi là O(V + E), do đi qua toàn bộ V Node và xem xét E cạnh
2. Inorder Traversal (LNR, Duyệt Trung Thứ Tự)?
⇨ Là 1 dạng của DFS
⇨ Khi có 1 Node con rút về là đưa Node cha vào mảng luôn
3. Preorder Traversal (NLR, Duyệt Tiền Thứ Tự)?
⇨ Là 1 dạng của DFS
⇨ Duyệt qua Node nào là đưa nó vào mảng luôn
4. Postorder Traversal (LRN, Duyệt Hậu Thứ Tự)?
⇨ Là 1 dạng của DFS
⇨ Node nào rút là chèn Node đó vào mảng luôn
5. Expression Tree?
⇨ Là 1 Ordered Rooted Tree mà tất cả Leaf Node là toán hạng và tất cả đỉnh trong là toán tử
⇨ Ví dụ



⇨ Khi viết biểu thức ứng với Tree này, ta viết theo 3 kiểu
⇨ Trung tố (Infix), dùng LNR, ta được mảng (x, +, y, ↑, 2, +, x, –, 4, /, 3), viết lại mảng này sử dụng ngoặc để nhóm, ta được ((x + y) ↑ 2) + ((x – 4) / 3)
⇨ Tiền tố (Prefix), dùng NLR, ta được mảng (+, ↑, +, x, y, 2, /, –, x, 4, 3), viết lại mảng này sử dụng ngoặc để nhóm, ta được +(↑(+(x, y), 2), /(–(x, 4), 3))
⇨ Hậu tố (Postfix), dùng LRN, ta được mảng (x, y, +, 2, ↑, x, 4, –, 3, /, +), viết lại mảng này sử dụng ngoặc để nhóm, ta được (((x, y)+, 2)↑, ((x, 4)–, 3)/) +
6. BFS (Breadth First Search, Tìm Kiếm Theo Chiều Rộng)?
⇨ Là cách duyệt 1 Graph theo kiểu phân tán từ điểm ban đầu, duyệt hết các Node cùng độ sâu trước

Path Finding – Tìm Đường:

1. Tìm Đường Đi Ngắn Nhất = Thuật Toán Dijkstra?

- ⇨ Cho Weighted Graph G bất kì với tất cả trọng số không âm và ta muốn tìm đường đi ngắn nhất tới tất cả các Node trong G từ Node A, giả sử từ A ta có thể đi đến mọi nơi trong G
- ⇨ Bước 1, đi tới từng Node, kí hiệu ∞ lên đầu chúng, giá trị này gọi là Value
- ⇨ Bước 2, đưa con trỏ trỏ vào Node A, thay Value của A thành 0_A, đánh dấu A, thay Value của các Node hàng xóm của A thành d_A = khoảng cách giữa A và chúng
- ⇨ Bước 3, duyệt qua toàn bộ các Node không được đánh dấu, đưa con trỏ trỏ vào Node có Value nhỏ nhất, gọi Node này là B, đánh dấu B
- ⇨ Bước 4, thay Value của các Node hàng xóm chưa được đánh dấu của B thành d_B = Value của B + khoảng cách giữa B và chúng, hoặc không thay nếu d_B > Value hiện tại của chúng
- ⇨ Bước 5, bắt đầu lại từ bước 3
- ⇨ Lặp đi lặp lại cho đến khi tất cả Node đều được đánh dấu, khi đó Value d_X của mỗi Node chính là độ dài quãng đường ngắn nhất từ Node A tới Node đó
- ⇨ Dễ thấy, gọi số Node là V, gọi số kết nối hay cạnh là E, do ta đi qua tất cả V Node, mỗi lần đi qua, ta lại phải tìm Node có giá trị nhỏ nhất trong V Node, đồng thời từ Node hiện tại, ta phải cập nhật Value của các Node hàng xóm thông qua các cạnh, nên thời gian thực thi của thuật toán này là $O(V^2 + E)$, nhưng E rất nhỏ so với V^2 nên thời gian thực thi là $O(V^2)$
- ⇨ Nếu tìm Min Value = cách sử dụng Min Heap, thì cứ mỗi lần cập nhật Value của Node hàng xóm, ta lại xóa và chèn vào Heap, dễ thấy tổng thời gian thực thi điều này là O(Elog(V)), mỗi lần tìm Min Value, ta xóa gốc của Heap, tổng thời gian thực thi điều này là O(Vlog(V)), như vậy toàn bộ thời gian thực thi nếu sử dụng Heap là O((V + E)log(V))
- ⇨ Cơ chế
- ⇨ Giữ trong đầu định lí sau, xét 1 đường đi nào đó từ A đến D, đường đi này được coi là ngắn nhất rồi, chọn điểm B và C trên đường đi này, thì đoạn đường từ B đến C trên đường đi này cũng chính là đường đi ngắn nhất từ B đến C
- ⇨ Minh họa
- ⇨ https://drive.google.com/file/d/1RzyUTcYoiOYRGpDRSj4s0iVZ5e0mr_s9/view?usp=sharing

2. Tìm Đường Nhanh = Thuật Toán A* (A Star)?
- ⇨ Đéo khác gì Dijkstra, chỉ có những sự khác biệt sau
- ⇨ Mục tiêu là tìm nhanh đường đi từ 1 Node A đến Node B nào đó, đường đi này càng ngắn càng tốt, ngắn nhất thì tuyệt vời
- ⇨ Thuật toán dừng lại khi đến Node B
- ⇨ Để làm được điều này, mỗi Node sẽ được gán thêm 1 giá trị Heuristic do chúng ta quy định, khi quyết định ta sẽ đưa con trỏ tới Node nào tiếp theo, thì không phải dựa vào Value của Node nào có giá trị nhỏ nhất, mà dựa vào giá trị Value + Heuristic của Node nào nhỏ nhất
- ⇨ Thông thường giá trị Heuristic sẽ được tính = khoảng cách vật lí từ 1 Node đến Node đích
- ⇨ Nếu giá trị Heuristic hợp lí, thì thuật toán đảm bảo tìm được đường đi ngắn nhất từ A đến B, nhưng nếu giá trị Heuristic như cặc, thì không chắc

3. Tìm Đường Đi Ngắn Nhất = Thuật Toán Bellman Ford Cho Graph Có Trọng Số Âm?

- ⇨ Cơ bản là khắc phục nhược điểm của Dijkstra khi không hoạt động khi có trọng số âm, bù lại thời gian thực thi lâu hơn
- ⇨ Xét trường hợp Graph tồn tại Loop âm, Loop âm là Loop mà khi bạn di chuyển dọc theo nó nhiều lần, thì giá trị của bạn sẽ giảm liên tục đến $-\infty$, như vậy, mọi Node bị Loop âm chĩa vào sẽ không có đường đi ngắn nhất tới nó
- ⇨ Thuật toán Bellman Ford vừa giúp tìm đường đường đi ngắn nhất từ 1 Node đến tất cả các Node còn lại, vừa xác định những Node không có đường đi ngắn nhất tới nó
- ⇨ Giả sử ta muốn tìm đường đi ngắn nhất từ Node A đến tất cả các Node còn lại, cho tằng Graph không có Loop âm
- ⇨ Bước 1, đi tới từng Node, kí hiệu ∞ lên đầu chúng, giá trị này gọi là Value, riêng Node A thì Value = 0_A
- ⇨ Bước 2, lần lượt lặp qua tất cả các Node, thứ tự bất kì, tại mỗi Node, cập nhật Value của tất cả các Node hàng xóm giống như ở Dijkstra, lặp lại bước này đúng V – 1 lần, V là số Node
- ⇨ Bây giờ, ta sẽ chứng minh tại sao phải lặp V – 1 lần
- ⇨ Tưởng tượng giống chiêu E của Ryze, mỗi lần lặp, giá trị khác ∞ sẽ dần được mở rộng, gọi đường đi ngắn nhất từ A đến Node Z nào đó là P, P có k cạnh, rõ ràng $k \leq V - 1$, ta có các Node dọc theo P theo thứ tự là A, B, C, …, Z, theo định lí được nêu ở thuật toán Dijkstra, thì đường đi A, B cũng là đường đi ngắn nhất từ A đến B, đường đi A, B, C cũng là đường đi ngắn nhất từ A đến C, …, do đó, dễ thấy sau lần lặp 1, ta xác định được chính xác ngay Value của B, sau lần lặp 2, ta xác định được chính xác ngay Value của C, …, và lần lặp thứ k, ta xác định được chính xác ngay Value của Z, điều này là do các Node này có Value nhỏ nhất, nên Value của chúng không thể bị thay thế khi những Node xung quanh cập nhật Value của Node hàng xóm, mà như đã nói, k có Max = V – 1, nên ta phải lặp V – 1 lần mới đảm bảo
- ⇨ Nếu muốn xác định các Node không có đường đi ngắn nhất tới nó, thì đơn giản ta chỉ cần tiếp tục thực hiện lại bước 2, chỉ thay đổi là nếu Value của 1 Node bị cập nhật thì Value của nó ngay lập tức trở thành $-\infty$, đến cuối cùng, những Node có Value là $-\infty$ chính là những Node cần tìm
- ⇨ Dễ thấy thời gian thực thi của thuật toán này là O(EV), E là số cạnh của Graph
- ⇨ Nhược điểm
- ⇨ Dễ thấy ta hoàn toàn có thể dùng Bellman Ford để tìm độ dài quãng đường ngắn nhất giữa 2 Node bất kì, = cách chọn từng Node làm gốc rồi thực hiện Bellman Ford, thời gian thực thi là $O(EV^2)$, tuy nhiên, với trường hợp Complete Graph, dễ thấy thời gian thực thi lên tới $O(V^4)$

4. Tìm Độ Dài Đường Đi Ngắn Nhất Giữa Tất Cả Các Cặp Node = Thuật Toán Floyd Warshall Cho Graph Có Trọng Số Âm?

- ⇨ Cơ bản khắc phục nhược điểm thời gian thực thi lâu vãi đái của Bellman Ford
- ⇨ Bước 1, tạo 1 ma trận M kích thước V x V, V là số Node, mục đích của ta là hàng I cột J của M phải chứa độ dài đường đi ngắn nhất từ Node thứ I đến Node thứ J
- ⇨ Bước 1, Fill M với giá trị là độ dài cạnh IJ tương ứng với hàng I cột J, nếu không tồn tại cạnh thì để ∞
- ⇨ Bước 2, chọn Node thứ 1 làm trung gian, so sánh giá trị hiện tại trong từng ô ở hàng I cột J với tổng độ dài cạnh I1 + cạnh 1J, nếu tổng này bé hơn thì thay nó vào ô

⇨ Bước 3, chọn Node thứ 2 làm trung gian, so sánh giá trị hiện tại trong từng ô ở hàng I cột J với tổng độ dài cạnh I2 + cạnh 2J, nếu tổng này bé hơn thì thay nó vào ô
⇨ Bước 4, …
⇨ Lặp đi lặp lại cho đến hết Node ta được ma trận cần tìm
⇨ Cơ chế
⇨ Dễ thấy bản chất ở cuối vòng lặp, xét tất cả các đường đi được cấu thành từ các Node thứ I, 1, 2, 3, …, V, J, với V là số Node, Node thứ I và Node thứ J lần lượt là đầu và đuôi của đường đi, ví dụ I, 1, 3, J, hay I, 5, 2, 1, J, …, khi đó ô hàng I cột J là độ dài đường đi nhỏ nhất trong những đường đi này
⇨ Dễ thấy thời gian thực thi thuật toán này là $O(V^3)$, nhanh hơn Bellman Ford
⇨ Nhược điểm
⇨ Dễ thấy Floyd Warshall không hoạt động nếu Graph có Loop âm

5. Tìm Lối Thoát Khỏi Ma Trận = Thuật Toán Random Mouse?

⇨ Là chọn 1 hướng ngẫu nhiên rồi đi thẳng, gặp vật cản thì ngay lập tức chọn hướng ngẫu nhiên khác, lặp đi lặp lại cho tới khi thoát, chậm vãi cả lồn

6. Tìm Đường Đi Ngắn Thứ K = Thuật Toán Yen?

⇨ Cho Weighted Graph G bất kì với tất cả trọng số không âm và ta muốn tìm đường đi ngắn thứ K từ Node A tới Node B, nghĩa là ta bây giờ không muốn tìm đường đi ngắn nhất nữa mà muốn tìm đường đi ngắn thứ 2, thứ 3, …
⇨ Thuật toán này sẽ bỏ qua các đường đi có Loop
⇨ Bước 1, tìm đường đi ngắn nhất từ A tới B bằng thuật toán nào đó, ví dụ Dijkstra, gọi đường đi này là P, tưởng tượng đường P như thân cây
⇨ Bước 2, lặp qua lần lượt các Node trong P, từ A đến ngay trước B, với mỗi Node C ta tìm đường đi ngắn nhất từ C tới B bằng thuật toán nào đấy, gọi đường này là P', với Graph tương ứng là G đã loại bỏ cạnh nối từ C tới Node ngay sau C trên P, đi 1 lèo từ A tới C, rồi từ C men theo P' đến B, ta được đường đi H, thêm H vào tập M là tập dùng để chứa các đường đi có khả năng trở thành đường đi ngắn thứ K, và M đang rỗng
⇨ Bước 3, chọn đường đi ngắn nhất trong M, ta được đường đi từ A tới B ngắn thứ 2, gọi đường này là P_2 đồng thời xóa đường này khỏi M
⇨ Bước 4, lặp qua lần lượt các Node trong P_2, từ A đến ngay trước B, với mỗi Node C ta tìm đường đi ngắn nhất từ C tới B bằng thuật toán nào đấy, gọi đường này là P', với Graph tương ứng là G đã loại bỏ cạnh nối từ C tới Node ngay sau C trên P_2 cùng với các cạnh trên P_1 mà có 1 đầu là C, lưu ý nếu trong M có đường mà nó chứa đoạn từ A tới C trên P_2 thì cũng xóa luôn cạnh nối từ C tới Node ngay sau C của đường đó, đi 1 lèo từ A tới C, rồi từ C men theo P' đến B, ta được đường đi H, thêm H vào tập M
⇨ Bước 5, chọn đường đi ngắn nhất trong M, ta được đường đi từ A tới B ngắn thứ 3, gọi đường này là P_3 đồng thời xóa đường này khỏi M
⇨ Bước 6, quay lại thực hiện bước 4 và 5 nhiều lần, thay chữ P_3 thành P_{i+1}, P_2 thành P_i, P_1 thành P_1, P_2, …, và P_{i-1}, i chạy từ 3 tới K – 1
⇨ Cuối cùng, ta sẽ được đường đi ngắn nhất, ngắn nhì, ngắn 3, …, ngắn thứ K

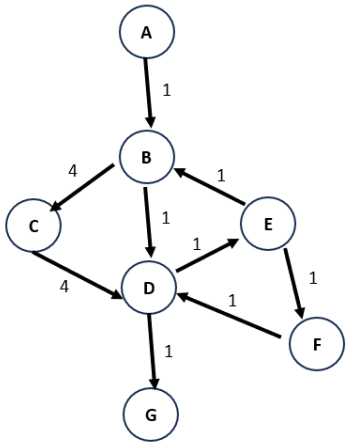
7. Tìm Đường Đi Ngắn Thứ K = Quy Hoạch Động?

⇨ Cho Weighted Graph G bất kì với tất cả trọng số không âm và ta muốn tìm đường đi ngắn thứ K từ Node A tới Node B
⇨ Khắc phục nhược điểm của thuật toán Yen là nó sẽ tính luôn Loop

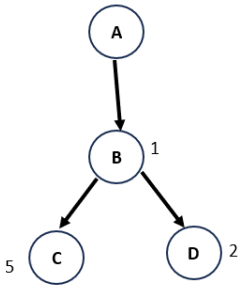
⇨ Cơ chế là biểu diễn G dưới dạng Tree, bắt đầu từ Node A, ta luôn xét Leaf Node mà đường đi tới nó hiện tại là ngắn nhất trong tất cả Leaf Node, không tính Leaf Node bị đánh dấu, nếu có nhiều đường đi ngắn nhất thì chọn ngẫu nhiên, từ Leaf Node đang xét chĩa ra các Node bị nó chĩa tới trong G, nếu trong các Node này có B thì đánh dấu lại, 1 Node trong G mà bạn đã khám phá từ nó K lần thì không được khám phá nữa, đánh dấu nó lại, dừng thuật toán khi Leaf Node G xuất hiện đúng K lần

⇨ Đọc đường đi của các Leaf Node B bị đánh dấu, ta được các đường đi ngắn nhất, ngắn nhì, ngắn 3, …, ngắn thứ K

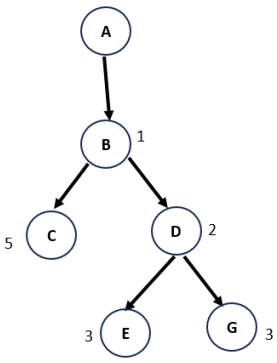
⇨ Ví dụ cho Graph sau, ta muốn tìm đường đi ngắn thứ 3 từ A tới G



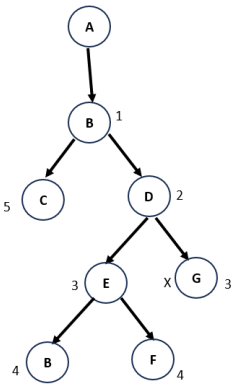
⇨ Vẽ Node A, sau đó chĩa vào B, sau đó từ B chĩa ra C và D



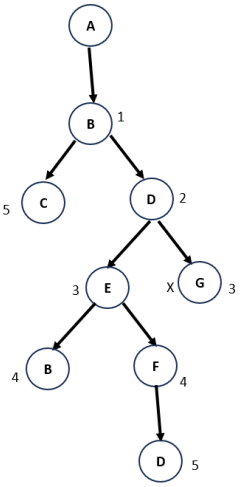
⇨ Do từ A tới D có độ dài = 2 là ngắn nhất, nên ta ưu tiên khám phá từ D



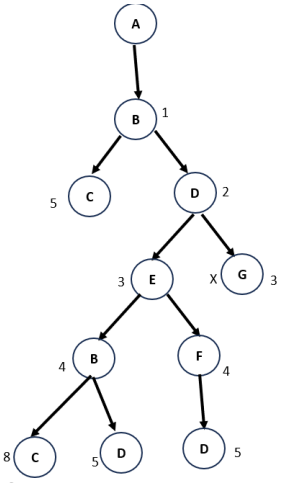
⇨ Ở đây cả E và G đều có quãng đường là 3 và là ngắn nhất, ta có thể chọn 1 trong 2 để tiếp tục khám phá, giả sử chọn E, lưu ý đánh dấu G



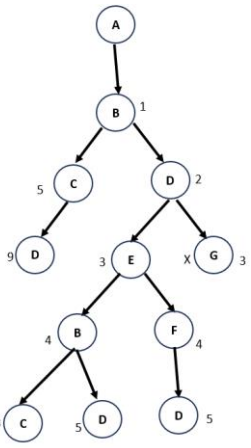
⇨ Ở đây cả Leaf Node B và F đều có quãng đường bằng 4 và là nhỏ nhất, ta chọn đại F để khám phá



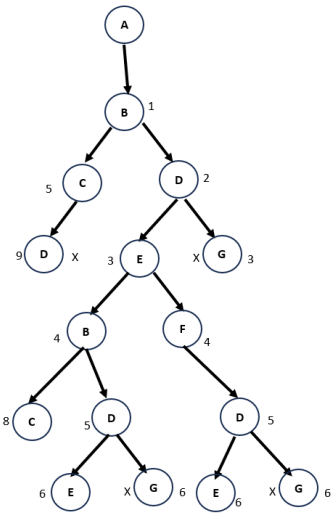
⇨ Ở đây Leaf Node B có quãng đường = 4 là ngắn nhất, nên khám phá từ nó



⇨ Ở đây, có cả 3 Leaf Node đều có quãng đường = 5 là nhỏ nhất, ta chọn đại C ở bên trên



⇨ Ở đây 2 Leaf Node D ở bên dưới có quãng đường = 5 là nhỏ nhất, chọn đại cái bên trái, tiếp tục làm tương tự các bước trên 1 lúc, ta được



⇨ Như vậy là đã có đủ 3 tuyến đường từ A tới G lần lượt là ngắn nhất, ngắn nhì, ngắn 3, để thấy từ D ta đã khám phá 3 lần, nên Leaf Node D bên trái bị đánh dấu

⇨ Tuyến đường ngắn nhất là ABDG, chiều dài 3

⇨ Tuyến đường ngắn nhì = ngắn 3 lần lượt là ABDEBDG và ABDEFDG, chiều dài = 6

SCC – Strongly Connected Component – Thành Phần Liên Thông Mạnh:

1. Thành Phần Liên Thông (Connected Component)?

⇨ Là 1 phần của Undirected Graph G, sao cho tồn tại ít nhất 1 đường đi từ 1 Node bất kì đến 1 Node bất kì khác trong phần này, đồng thời nó là Graph con đã to hết mức có thể, để không chứa trong Graph con khác

⇨ Bản chất, 1 thành phần liên thông có thể được coi là 1 Node, ta có thể biểu diễn Undirected Graph chỉ = tất cả thành phần liên thông của nó

⇨ 1 Node được gọi là đỉnh cắt (Cut Vertex) hoặc đỉnh khớp (Articulation Point) nếu loại bỏ nó ra khỏi Undirected Graph cùng với các cạnh có 1 đầu là nó, thì số thành phần liên thông của Graph này tăng lên

⇨ Nếu Connected Simple Graph chứa đỉnh cắt hoặc Node treo thì không tồn tại Hamilton Circuit
⇨ 1 cạnh được gọi là cạnh cắt (Cut Edge) hoặc cầu (Bridge) nếu loại bỏ nó ra khỏi Undirected Graph, vẫn giữ các 2 Node ở 2 đầu, thì số thành phần liên thông của Graph này tăng lên
⇨ 1 Connected Undirected Graph được gọi là đồ thị không thể phân tách (Nonseperable Graph) nếu nó không chứa đỉnh cắt
⇨ Số lượng Node ít nhất cần bị cắt để số lượng thành phần liên thông của Undirected Graph G tăng lên được gọi là liên thông đỉnh (Vertex Connectivity) của G, kí hiệu K(G)
⇨ Số lượng cạnh ít nhất cần bị loại bỏ để số lượng thành phần liên thông mạnh của Undirected Graph G tăng lên được gọi là liên thông cạnh (Edge Connectivity) của G, kí hiệu λ(G)
2. Thành Phần Liên Thông Mạnh (Strongly Connected Component)?
⇨ Tương đương định nghĩa của thành phần liên thông, chỉ thay Undirected Graph thành Directed Graph
3. Thành Phần Liên Thông Yếu (Weakly Connected Component)?
⇨ Là 1 phần của Directed Graph G, không phải thành phần liên thông mạnh, sao cho sau khi thế các cạnh có hướng thành cạnh vô hướng trong phần này, ta được 1 thành phần liên thông, đồng thời nó là Graph con đã to hết mức có thể, để không chứa trong Graph con khác
4. Tìm Thành Phần Liên Thông Mạnh = Thuật Toán Kosaraju?
⇨ Mục đích là từ 1 Graph, ta muốn liệt kê tất cả thành phần liên thông mạnh của nó, mỗi thành phần liên thông mạnh ta cần biết những Node nào nằm trong đó
⇨ Bước 1, tạo Stack, chọn 1 Node chưa đi qua làm điểm xuất phát, tiến hành DFS từ Node đó, mỗi lần hoàn tất DFS trên 1 Node thì đưa Node đó vào Stack
⇨ Bước 2, lặp lại bước 1 cho đến khi tất cả các Node đều được đi qua, lưu ý không đi qua Node đã đi qua rồi
⇨ Bước 3, đảo chiều tất cả các cạnh của Graph
⇨ Bước 4, Pop Node trên cùng của Stack, tiến hành DFS từ Node đó trên Graph đã bị đổi chiều, tất cả những Node bị đi qua được gôm chung thành 1 thành phần liên thông mạnh,
⇨ Bước 5, lặp lại bước 4 cho tới khi Stack trống, lưu ý, không DFS trên những Node đã đi qua rồi
⇨ Bản chất thuật toán này dựa vào tính chất thành phần liên thông mạnh dù có đổi chiều tất cả các cạnh thì vẫn là thành phần liên thông mạnh
⇨ Minh họa
⇨ https://drive.google.com/file/d/1rg4Xn4bRbASv5aRuqPwfT2bAIgTfC40z/view?usp=sharing
⇨ Do toàn bộ là DFS nên thời gian thực thi thuật toán này là O(V + E)
⇨ Nhược điểm
⇨ Phải thực hiện DFS tới 2 lần
5. Tìm Thành Phần Liên Thông Mạnh = Thuật Toán Tarjan?
⇨ Cơ bản khắc phục việc phải tiến hành 2 lần DFS của Kosaraju
⇨ Bước 1, chọn 1 Node chưa đi qua làm điểm xuất phát, tiến hành DFS từ Node đó, đặt Index cho mỗi Node đi qua, đồng thời cũng đặt Value = Index, Index tăng dần, nếu va chạm lại đường đi, nói cách khác tưởng tượng Game rắn săn mồi đầu con rắn cắn vào thân nó, thì cập nhật Value của đầu rắn = Index chỗ nó cắn

nếu Index đó < Value hiện tại, khi lan truyền ngược, cập nhật Value của các Node thành Value của Node nó chĩa vào, nếu Value của Node nó chĩa vào < Value hiện tại của nó, nếu khi lan truyền ngược, bắt gặp 1 Node có Value = Index của nó, thì cắt nhánh tương ứng với Node đó, nhánh đó chính là 1 SCC

- ⇨ Bước 2, lặp lại bước 1 tại Node xuất phát khác chưa đi qua cho đến khi tất cả các Node đều được đi qua, không đi lại các DFS trước
- ⇨ Bản chất là lợi dụng tính chất 1 DFS nếu chứa 1 Node của 1 SCC thì nó chứa nguyên SCC đó
- ⇨ Minh họa
- ⇨ https://drive.google.com/file/d/14OTR9eE4TPdTgaIsBhvC3uyd2ecb0OMI/view?usp=drive_link
- ⇨ Dễ thấy do sử dụng DFS nên thời gian thực thi của thuật toán này là O(V + E), V là số Node, E là số cạnh, và ta chỉ cần thực hiện 1 lần DFS trên toàn bộ Graph

Sorting – Sắp Xếp:

1. Bubble Sort (Sắp Xếp Nổi Bọt)?

- ⇨ Là kiểu sắp xếp mà mỗi lần lặp giá trị lớn nhất sẽ được đẩy xuống cuối, gọi là nổi bọt
- ⇨ Bước 1, chạy từ đầu đến cuối, 2 phần tử kề nhau không đúng thứ tự thì hoán vị
- ⇨ Bước 2, chạy từ đầu đến kế cuối, 2 phần tử kề nhau không đúng thứ tự thì hoán vị
- ⇨ Bước 3, …
- ⇨ Lặp đi lặp lại cho đến khi điểm xuất phát và dừng lại trùng nhau
- ⇨ Ví dụ

Lần hoán vị	
1	[9, 7, 8, 4]
2	[7, 9, 8, 4]
3	[7, 8, 9, 4]
4	[7, 8, 4, 9]
5	[7, 8, 4, 9]
6	[7, 4, 8, 9]
7	[4, 7, 8, 9]

2. Insertion Sort (Sắp Xếp Chèn)?

- ⇨ Là kiểu sắp xếp mà ta chèn phần tử tiếp theo vào dãy đã sắp xếp đằng trước
- ⇨ Bước 1, hoán vị phần tử thứ 2 với thứ nhất nếu không đúng thứ tự
- ⇨ Bước 2, cầm phần tử thứ 3 ra, chèn vào 2 phần tử đằng trước theo đúng thứ tự
- ⇨ Bước 3, cầm phần tử thứ 4 ra, chèn vào 3 phần tử đằng trước theo đúng thứ tự
- ⇨ Bước 4, …
- ⇨ Lặp đi lặp lại cho đến khi chèn phần tử cuối cùng
- ⇨ Ví dụ

Lần chèn	
1	[9, 7, 8, 4]
2	[7, 9, 8, 4]
3	[7, 8, 9, 4]
4	[4, 7, 8, 9]

3. Selection Sort (Sắp Xếp Lựa Chọn)?

⇨ Là kiểu sắp xếp mà mỗi lần lặp, ta tìm Min của dãy còn lại rồi hoán vị nó với phần tử đầu tiên
⇨ Bước 1, tìm Min của toàn mảng, hoán vị nó với phần tử đầu tiên
⇨ Bước 2, tìm Min từ phần tử thứ 2 đến hết, hoán vị nó với phần tử thứ 2
⇨ Bước 3, tìm Min từ phần tử thứ 3 đến hết, hoán vị nó với phần tử thứ 3
⇨ Bước 4, …
⇨ Lặp đi lặp lại cho đến khi chạm đến cuối mảng
⇨ Ví dụ

Lần hoán vị	
1	[9, 8, 7, 4]
2	[4, 8, 7, 9]
3	[4, 7, 8, 9]
4	[4, 7, 8, 9]

4. Merge Sort (Sắp Xếp Trộn)?
⇨ Là kiểu sắp xếp chia để trị
⇨ Bước 1, cứ 2 phần tử một sắp xếp lại cho đúng thứ tự
⇨ Bước 2, cứ 4 phần tử một sắp xếp lại cho đúng thứ tự
⇨ Bước 3, cứ 8 phần tử một sắp xếp lại cho đúng thứ tự
⇨ Bước 4, …
⇨ Lặp đi lặp lại cho đến khi vượt quá kích thước mảng
⇨ Việc sắp xếp lại cho đúng thứ tự diễn ra khá đơn giản, xét trường hợp 8 phần tử một sắp xếp lại cho đúng thứ tự, thì ta đang nói đến 2 mảng liên tục có 4 phần tử, 2 mảng này đã được sắp xếp theo đúng thứ tự trước đó, ví dụ 2 mảng đó là

[4, 8, 9, 15]	[1, 2, 3, 20]

⇨ Đầu tiên, so sánh 4 với 1, thấy 1 bé hơn, chọn 1 rồi liên tục chọn 2, 3, cho đến khi gặp 1 phần tử > 8, ta đổi bên, chọn liên tục 8, 9, 15, cho đến khi hết mảng hoặc gặp phần tử > 20, tiếp tục đổi bên, chọn 20
⇨ Ví dụ Full Merge Sort

Lần sắp xếp	
1	[9, 8, 7, 4, 1, 5, 15, 10]
2	[8, 9, 4, 7, 1, 5, 10, 15]
3	[4, 7, 8, 9, 1, 5, 10, 15]
4	[1, 4, 5, 7, 8, 9, 10, 15]

⇨ Dễ thấy Merge Sort có thời gian thực thi là O(nlog(n))
5. Quick Sort (Sắp Xếp Nhanh)?
⇨ Cũng là kiểu sắp xếp chia để trị, nhưng dùng trụ cột
⇨ Bước 1, chọn trụ cột là 1 phần tử tùy ý có thể là phần tử chính giữa, cuối, đầu hay cái đéo gì cũng được
⇨ Bước 2, hoán vị trụ cột với phần tử cuối cùng, xét dãy con từ phần tử đầu tiên đến kế cuối, đặt 1 con trỏ bên trái, dò từ bên trái sang cho đến khi gặp phần tử > trụ cột, đặt 1 con trỏ bên phải, dò từ bên phải sang cho đến khi gặp phần tử < trụ cột, hoán vị 2 phần tử này, rồi tiếp tục dò tiếp cho đến khi con trỏ bên trái đứng bên phải con trỏ bên phải thì dừng, rồi hoán vị con trỏ bên trái với trụ cột, khi này dễ thấy bên phải trụ cột toàn là phần tử lớn hơn nó, còn bên trái toàn phần tử nhỏ hơn nó, nói cách khác, trụ cột đã ở đúng vị trí, nghĩa là đây cũng là vị trí cuối cùng của nó
⇨ Bước 3, tiếp tục làm như vậy với phần mảng bên phải của trụ cột và phần mảng bên trái của trụ cột 1 cách đệ quy cho đến khi không thể chia mảng được nữa
⇨ Ví dụ, ta sẽ mặc định chọn phần tử đầu tiên làm trụ cột

Lần hoán vị	
1	[9, 8, 7, 4, 1, 5, 15, 10]
2	[10, 8, 7, 4, 1, 5, 15, 9]
3	[5, 8, 7, 4, 1, 10, 15, 9]
4	[5, 8, 7, 4, 1, 9, 15, 10]
5	[1, 8, 7, 4, 5, 9, 10, 15]
6	[1, 4, 5, 8, 7, 9, 10, 15]
7	[1, 4, 5, 7, 8, 9, 10, 15]

⇨ Dễ thấy trong trường hợp ngon nhất, thì Quick Sort na ná Merge Sort, do đó thời gian thực thi là O(nlog(n)), nhưng trong trường hợp tồi tệ nhất, bạn chọn trụ cột thế đéo nào đó mà khi quét 2 con trỏ, đéo có hoán vị nào xảy ra, dẫn đến trụ cột vẫn ở cuối, dễ thấy trong trường hợp này thời gian thực thi là $O(n^2)$

6. Radix Sort (Sắp Xếp Theo Cơ Số)?

⇨ Là kiểu sắp xếp theo hàng đơn vị, rồi hằng chục, hàng trăm, …

⇨ Bước 1, chỉ quan tâm chữ số hàng đơn vị của các phần tử và sắp xếp theo chúng, sắp xếp = cách tạo ra 10 hộp có nhãn 0, 1, 2, …, 9, rồi đặt các phần tử vào hộp tương ứng với chữ số hàng đơn vị của nó

⇨ Bước 2, nối 10 hộp lại, được mảng mới, tiếp tục lặp lại bước 1 nhưng với chữ số hàng chục

⇨ Bước 3, như bước 2, nhưng với chữ số hàng trăm

⇨ Bước 4, …

⇨ Lặp đi lặp lại cho đến khi không còn chữ số

⇨ Ví dụ

Sắp xếp theo	
Chưa sắp xếp	[899, 18, 27, 42, 1, 93, 145, 100]
Hàng đơn vị	[100, 1, 42, 93, 145, 27, 18, 899]
Hàng chục	[100, 1, 18, 27, 42, 145, 93, 899]
Hàng trăm	[1, 18, 27, 42, 93, 100, 145, 899]

⇨ Dễ thấy Radix Sort có thời gian thực thi là O(d(n + b)) với d là số chữ số thập phân, ví dụ đến hàng trăm thì d = 3, n là kích thước mảng, b là số hộp, ở đây là 10, do ta tính cả thời gian tạo hộp

7. Bogo Sort?

⇨ Joke, sắp xếp ngẫu nhiên mảng liên tục cho đến khi nó sắp xếp đúng

8. Heap Sort?

⇨ Sắp xếp = cấu trúc Heap

⇨ Bước 1, từ mảng ban đầu tạo Max Heap tương ứng = cách chèn từng phần tử vào Heap

⇨ Bước 2, xóa Node gốc của Heap được Heap mới, đưa Node vừa xóa vào cuối Heap mới và kệ mẹ nó

⇨ Bước 3, xóa Node gốc của Heap mới được Heap mới hơn, đưa Node vừa xóa vào cuối Heap mới hơn và kệ mẹ nó

⇨ Bước 4, …

⇨ Lặp đi lặp lại cho đến khi xóa hết Node

⇨ Ví dụ

Mảng ban đầu	[899, 18, 27, 42, 1, 93, 145, 100]
Max Heap	[899, 100, 145, 42, 1, 27, 93, 18]
Lần xóa 1	[145, 100, 93, 42, 1, 27, 18, 899]
Lần xóa 2	[100, 42, 93, 18, 1, 27, 145, 899]
Lần xóa 3	[93, 42, 27, 18, 1, 100, 145, 899]

Lần xóa 4	[42, 18, 27, 1, 93, 100, 145, 899]
Lần xóa 5	[27, 18, 1, 42, 93, 100, 145, 899]
Lần xóa 6	[18, 1, 27, 42, 93, 100, 145, 899]
Lần xóa 7	[1, 18, 27, 42, 93, 100, 145, 899]

⇨ Dễ thấy Heap Sort có thời gian thực thi là O(nlog(n))

9. Topological Sort?

⇨ Cho 1 DAG, bài toán đặt ra là tạo ra 1 mảng chứa tất cả các Node trong DAG này sao cho Node cha phải nằm phía bên trái Node con

⇨ Tưởng tượng để học được 1 môn, ta cần kiến thức từ những môn khác, để học những môn khác đó lại phải cần học môn khác nữa, …, Topological Sort sẽ giúp ta vạch ra lộ trình học

⇨ Bước 1, tạo mảng kích thước V, V là số Node

⇨ Bước 2, thực hiện DFS từ 1 Node bất kì chưa đi qua, sau khi hoàn tất DFS tại 1 Node thì đưa Node đó vào cuối mảng, nghĩa là ở vị trí xa nhất chưa bị chiếm

⇨ Bước 3, lặp lại bước 2 cho đến khi hết Node

⇨ Ta cũng có thể sử dụng BFS để Topological Sort, nhưng sẽ tốn thêm bộ nhớ

Error Detection – Phát Hiện Lỗi:

1. Mã Hamming?

⇨ Giả sử A muốn gửi 1 chuỗi Bit cho B, nhưng do sự cố 1 số Bit bị thay đổi, khi này sẽ có 1 số Bit được A thêm vào chuỗi ban đầu để bằng 1 cách nào đó, khi B nhận được chuỗi Bit tổng hợp này, sẽ có cách phát hiện Bit sai lệnh và sửa nó

⇨ Gọi chuỗi Bit ban đầu là S, và cho 1 số nguyên r từ 2 trở lên, khi này ta có thể chia S thành từng khúc, mỗi khúc gồm $2^r – r – 1$ Bit, rồi chèn r Bit kiểm lỗi ở các vị trí thích hợp trong khúc, như vậy độ dài tổng cộng của mỗi khúc lúc này là $2^r – 1$, ví dụ nếu r = 4, thì chia S thành từng khúc dài 11 Bit, rồi chèn 4 Bit ở các vị trí thích hợp trong khúc

⇨ Mã Hamming có thể mở rộng ra bằng cách chèn thêm 1 Bit Parity nữa vào đầu từng khúc, như vậy độ dài tổng cộng của mỗi khúc sẽ là 2^r

⇨ Kí hiệu một loại mã Hamming là

(<Độ Dài Tổng Cộng Mỗi Khúc>, <Độ Dài Ban Đầu Mỗi Khúc>)

⇨ Ví dụ nếu r = 4, và không mở rộng, thì kí hiệu là mã Hamming (15, 11)

⇨ Khoảng cách Hamming của một loại mã Hamming là số lượng Bit tối thiểu khác 0 bị thay đổi giá trị trong 1 khúc để ta không phát hiện lỗi, ví dụ khi r = 1, và không mở rộng, thì độ dài khúc tổng cộng = 3, ta phải thay đổi đồng thời giá trị của cả 3 Bit thì người nhận mới không phát hiện lỗi, do đó khoảng cách Hamming của mã này = 3

⇨ Xét trường hợp không mở rộng, thuật toán chi tiết để thêm Bit vào từng khúc của S như sau

⇨ Bước 1, chia S thành các khúc dài $2^r – r – 1$ Bit, khúc cuối cùng có thể có ít Bit hơn

⇨ Bước 2, chèn vào cuối mỗi khúc r ô trống, được các khúc lớn

⇨ Bước 3, đánh Index các ô Bit trong mỗi khúc lớn, Bit đầu Index 1, và cứ thế tăng dần và Reset khi qua khúc mới

⇨ Bước 4, lặp qua từng ô Bit của từng khúc, mỗi khi gặp ô Bit có Index là lũy thừa của 2 như 1, 2, 4, 8, … thì đẩy các Bit từ ô này trở đi trong khúc tương ứng sang bên phải 1 ô, như vậy khi này các ô lũy thừa 2 này sẽ trống

⇨ Bước 5, thực hiện phép EX – OR trên toàn bộ Index của các ô có giá trị = 1, đổi Index thành nhị phân r Bit trước khi EX – OR, kết quả là 1 chuỗi r Bit, gọi là R, đánh Index cho các Bit của R từ phải sang là 0, 1, 2, …, bây giờ điền giá trị vào các ô lũy thừa, giả sử ô này có Index 2^n, thì điền giá trị của Bit có Index n trong R vào

⇨ Ví dụ, cho S = 10110111101011111111010110111011101011010111010, r = 5

⇨ Bước 1, chia S thành các khúc dài 26 Bit

⇨ 10110111101011111111010110 và 11101110101101010111010

⇨ Bước 2 và 3, chèn 5 ô trống vào cuối mỗi khúc, đồng thời đánh Index

1	0	1	1	0	1	1	1	1	0	1	0	1	1	1	1	1	1	1	1	0	1	0	1	1	0					
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

⇨ Và

1	1	1	0	1	1	1	0	1	0	1	1	0	1	0	1	1	1	0	1	0					
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26

⇨ Bước 4, đẩy Bit để dọn chỗ cho Bit lũy thừa

		1		0	1	1		0	1	1	1	1	0	1		0	1	1	1	1	1	1	1	1	0	1	0	1	1	0
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

⇨ Và

		1		1	1	0		1	1	1	0	1	0	1		1	0	1	0	1	1	1	0	1	0
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26

⇨ Bước 5, EX – OR Index của các ô có giá trị 1

⇨ Khúc 1

$$3 \oplus 6 \oplus 7 \oplus 10 \oplus 11 \oplus 12 \oplus 13 \oplus 15 \oplus 18 \oplus 19 \oplus 20 \oplus 21 \oplus 22 \oplus 23 \oplus 24 \oplus 25 \oplus 27 \oplus 29 \oplus 30 = 21 = 10101$$

⇨ Như vậy điền 1, 0, 1, 0, 1 lần lượt vào ô Index 1, 2, 4, 8, 16

1	0	1	1	0	1	1	0	0	1	1	1	1	0	1	1	0	1	1	1	1	1	1	1	1	0	1	0	1	1	0
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

⇨ Khúc 2

$$3 \oplus 5 \oplus 6 \oplus 9 \oplus 10 \oplus 11 \oplus 13 \oplus 15 \oplus 17 \oplus 19 \oplus 21 \oplus 22 \oplus 23 \oplus 25 = 5 = 00101$$

⇨ Như vậy điền 1, 0, 1, 0, 0 lần lượt vào ô Index 1, 2, 4, 8, 16

1	0	1	1	1	1	0	0	1	1	1	0	1	0	1	0	1	0	1	0	1	1	1	0	1	0
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26

⇨ Cuối cùng, nối các khúc này lại, ta được chuỗi Bit hoàn thiện để gửi

S’ = 10110110011110110111111110101101011110011101010101010111010

⇨ Để giải mã đoạn Code này, dùng thuật toán sau, xét trường hợp không mở rộng, S’ là đoạn mã nhận được

⇨ Bước 1, chia S' thành các khúc dài $2^r - 1$ Bit, khúc cuối cùng có thể có ít Bit hơn, đánh Index tương tự như thuật toán trước

⇨ Bước 2, với từng khúc, thực hiện phép EX – OR trên toàn bộ Index của các ô có giá trị = 1, nếu kết quả = 0 thì coi như khúc này không có lỗi, nếu kết quả khác 0, ví dụ = n, thì ô có Index n có lỗi, ta Flip giá trị ô này, ô lỗi có thể là ô lũy thừa vẫn được

⇨ Bước 3, loại bỏ các ô lũy thừa rồi nối lại các khúc, ta được chuỗi S ban đầu

⇨ Ví dụ, S' ta đã có ở ví dụ trước với Bit màu đỏ bị lỗi Flip, r = 5

S' = 10110110011111**1**1011111111101011010111100111010101010111010

⇨ Bước 1, chia S' thành các khúc dài 31 Bit và đánh Index

1	0	1	1	0	1	1	0	0	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	0	1	0	1	1	0
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

⇨ Và

1	0	1	1	1	1	0	0	1	1	1	0	1	0	1	0	1	0	1	0	1	1	1	0	1	0
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26

⇨ Bước 2, EX – OR Index của các ô có giá trị = 1

⇨ Khúc 1

$$1 \oplus 3 \oplus 4 \oplus 6 \oplus 7 \oplus 10 \oplus 11 \oplus 12 \oplus 13 \oplus 14 \oplus 15 \oplus 16 \oplus 18 \oplus 19 \oplus 20 \oplus 21 \oplus 22 \oplus 23 \oplus 24 \oplus 25 \oplus 27 \oplus 29 \oplus 30 = 14$$

⇨ Như vậy ô 14 lỗi, ta Flip giá trị của nó từ 1 xuống 0, được

10110110011111**0**11011111111010110

⇨ Khúc 2

$$1 \oplus 3 \oplus 4 \oplus 5 \oplus 6 \oplus 9 \oplus 10 \oplus 11 \oplus 13 \oplus 15 \oplus 17 \oplus 19 \oplus 21 \oplus 22 \oplus 23 \oplus 25 = 0$$

⇨ Như vậy khúc 2 không có lỗi, giữ nguyên nó

10111100111010101010111010

⇨ Bước 3, loại bỏ ô lũy thừa

⇨ Khúc 1

10110111101011111111010110

⇨ Khúc 2

11101110101101010111010

⇨ Nối lại

10110111101011111111010110111011101011010111010

⇨ Dễ thấy chuỗi này y chang S

2. Mã Hamming (7, 4)?

⇨ Ta có ma trận tạo mã

$$G = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}$$

⇨ Ma trận Check lỗi

$$H = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}$$

⇨ Ma trận giải mã

$$R = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

⇨ Cho S là 1 chuỗi 4 Bit, chuyển nó thành 1 Vector, hay ma trận 1 cột, chiều đọc từ trái sang phải ứng với từ trên xuống dưới

⇨ Khi này chuỗi S' sẽ được tính bằng công thức nhân ma trận sau

$$S' = (G^T S)\ mod\ 2$$

⇨ Gọi I là Index dạng nhị phân của Bit bị lỗi trong S', I là Vector, hay ma trận 1 cột, MSB nằm dưới cùng

$$I = (HS')\ mod\ 2$$

⇨ Để trả về chuỗi ban đầu

$$S = RS'$$

⇨

Turing Machine:

Def:

Máy tính thay cho công việc lặp đi lặp lại dựa theo một hướng dẫn nào đó

Input là một chuỗi nhị phân, có một con mắt đang ở vị trí khởi đầu

Con mắt sẽ dựa vào State hiện tại và số nhìn thấy là 0 hay 1 để xóa, sửa, thêm 0 hoặc 1 rồi di chuyển đến vị trí nhất định và chuyển sang State mới

Khi tiến tới State nào đó sẽ dừng lại và xuất Output

Có thể thực hiện được mọi thuật toán

Halting Problem:

Def:

Không có máy tính nào có thể kiểm tra một chương trình sẽ dừng lại hay chạy mãi mãi

Proof:

Giả sử tồn tại máy 1 có thể cho biết một chương trình sẽ dừng lại hay chạy mãi mãi

Gọi máy 2 cho kết quả ngược lại máy 1

Giả sử máy 1 cho rằng chương trình sẽ dừng lại, thì máy 2 sẽ chạy liên tục, lấy máy 2 làm Input cho máy 1, máy 1 sẽ chạy liên tục, nên máy 2 dừng lại, mẫu thuẫn

Morse Code:

Def:

Mã hóa kí tự thành các chuỗi tín hiệu dài ngắn ngắt quãng khác nhau

Enigma Machine:

Def:

Máy mã hóa tin nhắn thành chuỗi văn bản khác

Kí tự giống nhau có thể cho ra kí tự khác nhau nên cực khó để giải mã

Bombe:

Def:

Máy giải mã Enigma bằng Brute Force nhưng tận dụng những yếu tố như việc một kí tự không thể cho ra chính nó

Monte Carlo:

Def:

Cho chạy Simulation Random, khi hết Game thì tổng hợp nước đi rồi Optimize

Partial Observable:

Def:

State còn thiếu một số thuộc tính của môi trường

Entropy: Độ ngẫu nhiên của một biến ngẫu nhiên x

Formula: $\sum_i -p_i log p_i = \int -p(x) log p(x) dx$

Reparameterization Trick: Biểu hiện normal distribution dưới dạng mu + sigma * e để backprop cho mu và sigma vì random sample không backprop được

Off-Policy: Hành động theo Policy này nhưng học theo Policy khác

Ex: DQN hành động theo E-Greedy Policy (chọn hành động ngẫu nhiên xen kẽ với chọn hành động để Max Q-Value) nhưng học theo Greedy Policy (chọn hành động để Max Q-Value)

DDPG hành động theo Policy hiện tại nhưng học theo những Policy trước đó (Sample hành động từ Experience Replay trong quá khứ)

On-Policy: Hành động và học theo cùng một Policy

Ex: SARSA cùng hành động và học theo E-Greedy Policy

Offline-Learning: Học khi có đủ dataset

Online-Learning: Học ngay khi data mới xuất hiện

Stochastic Policy: Policy mà tại state s bất kì, action sẽ được chọn theo tỉ lệ

Ex: Tại state s, 30% sang phải, 70% sang trái

Deterministic Policy: Policy mà tại state s bất kì, action chọn là chắc chắn

Ex: Tại state s, 100% sang phải

Policy Based:

Def:

Map thẳng State vào Action cụ thể

Ex:

$$\pi(s_3) = [1.2 \quad 2.5 \quad 3]$$

Value Based:

Def:

Không Store Policy

Học Value Function và chọn Action theo có Max Value

Variance: Phương sai

Formula:

$$Var(x) = \int (x - \bar{x})^2 f(x) dx = \sum_{i=1}^{n} \frac{(x_i - \bar{x})^2}{n} = \left(\sum_{i=1}^{n} \frac{x_i^2}{n} \right) - \bar{x}^2$$

$$Var(\bar{x}) = \frac{\sigma^2}{n} = E(\bar{x}^2) - \bar{x}$$

Meaning: Mỗi lần lấy n sample, tính mean, sau khi lấy nhiều lần, mean là một biến ngẫu nhiên và công thức trên tính variance của mean

Estimate:

$$Var(x) = \sum_{i=1}^{n} \frac{(x_i - \bar{x})^2}{n-1}$$

Covariance: Hiệp phương sai

Estimate: $\sum_{i=1}^{n} \frac{(x_i - \bar{x})(y_i - \bar{y})}{n-1}$

Meaning:

Variance theo 2 biến ngẫu nhiên

Covariance của 1 biến ngẫu nhiên với chính nó là variance

Covariance > 0 => x tăng y tăng, < 0 => x tăng y giảm, = 0, x tăng y const

Covariance Matrix: Matrix mà các entries là covariance của các biến ngẫu nhiên

Form:

$$\begin{matrix} Var(x) & Cov(x,y) & \dots \\ Cov(y,x) & Var(y) & \dots \\ \dots & \dots & \dots \end{matrix}$$

Multivariate Normal Distribution: Normal Distribution đa biến

Formula:

$$f(x) = \frac{1}{\sqrt{2\pi}^D \sqrt{|\Sigma|}} e^{-\frac{1}{2}(x-\bar{x})^T \Sigma^{-1} (x-\bar{x})}$$

Note:

D: Số biến ngẫu nhiên

Σ: Covariance Matrix

Proof: Đây là hàm Normal Distribution

Đặt $y = x - \bar{x}$

$$\int e^{-\frac{1}{2}(x-\bar{x})^T \Sigma^{-1} (x-\bar{x})} dx = \int e^{-\frac{1}{2} y^T \Sigma^{-1} y} dx$$

Σ symmetric => $\Sigma = U^T S U$

U là Orthogonal Matrix = Eigen Vector của Σ

S là Diagonal Matrix = Eigen Value của Σ

Đặt $z = Uy$

$$y^T \Sigma^{-1} y = y^T U^T S^{-1} U y = z^T S^{-1} z$$

$$|\Sigma| = |U^T S U| = |U^T U||S| = |S|$$

Mà S^{-1} là Diagonal Matrix => $e^{-\frac{1}{2}z^T S^{-1} z}$ kiểu Standard MNP =>

$e^{-\frac{1}{2}z^T S^{-1} z} = \sqrt{2\pi}^D \sqrt{|S|} = \sqrt{2\pi}^D \sqrt{|\Sigma|}$ => đpcm

Expected Value: Giá trị trung bình nhận được khi thực hiện hành động nào đó

Formula: E(X) = Integral(xL(X = x)dx)

Fisher Information: Cho một sample, trả về lượng thông tin mà sample cung cấp về biến ẩn

Formula: I(theta) = Integral(ln(L(theta, x)'theta ** 2 * L(theta, x))

KL Divergence (Kullback-Leibler Divergence): Khoảng cách giữa 2 probability distribution

(không symmetry, D(p, q) <> D(q, p))

Algo: tỉ lệ xuất hiện sequence này của p so với q, p (sequence) / q(sequence), normalize bằng số lần chọn(lấy căn n), lấy log

Formula: p1*log(p1/q1) + p2*log(p2/q2) + . + pn*log(pn/qn)

Ex:

P1: 0.2 P2: 0.8

Q1: 0.6 Q2: 0.4

~ 0.335

Maximum Entropy: Max -log(X)

Maximum Likelihood: Tìm distribution (normal, gamma, …) fit với data nhất

Max Log Likelihood: Thuật toán tìm mean, std của data:

Algo: L(mean, std | x1->xn) = L(mean, std | x1) *..* L(mean, std | xn)

Lấy log cho dễ lấy đạo hàm(log có peak tương tự)

Formula: Mean = mean x1 -> x2, std = sqrt(tổng square (x - mean) / n)

Hessian Matrix: Ma trận second derivative của multivariables function

Formula: $H[i, j] = f_{x[i]x[j]}^{(2)}$

Quadratic Approximation: Ước hệ hyperparalbol tại điểm (hơn ∇ là linear)

Formula: $f^{\sim}(x) = f(c) + \nabla(c)^T @ (x - c) + (x - c)^T @ \text{Hessian}(c) @ (x - c)$

Markov Property – Markovian:

Def:

State tiếp theo chỉ dựa vào State hiện tại

Transition Matrix:

Formula:

$$P = \begin{bmatrix} p(s_0|s_0) & p(s_0|s_1) & \dots & p(s_0|s_n) \\ p(s_1|s_0) & p(s_1|s_1) & \dots & p(s_1|s_n) \\ \dots & \dots & \dots & \dots \\ p(s_n|s_0) & p(s_n|s_1) & \dots & p(s_n|s_n) \end{bmatrix}$$

Tỉ lệ đạt State s_b từ State s_a sau k steps được cho bởi Matrix

$P(k) = P^k$

Emission Matrix:

Tỉ lệ đạt Observable State s' từ Hidden State s

Formula:

$$E = \begin{bmatrix} p(s'_0|s_0) & p(s'_0|s_1) & \dots & p(s'_0|s_n) \\ p(s'_1|s_0) & p(s'_1|s_1) & \dots & p(s'_1|s_n) \\ \dots & \dots & \dots & \dots \\ p(s'_m|s_0) & p(s'_m|s_1) & \dots & p(s'_m|s_n) \end{bmatrix}$$

Stationary Distribution:

Def:

Distribution X mà khi nhân với Transition Matrix P sẽ lại được Distribution đó

X là Eigen Vector của P với Eigen Value $= 1$

$PX = X$

Ex:

$$\begin{bmatrix} 0.3 & 0.1 & 0.5 \\ 0.5 & 0.8 & 0.4 \\ 0.2 & 0.1 & 0.1 \end{bmatrix} \begin{bmatrix} \frac{7}{38} \\ \frac{53}{76} \\ \frac{9}{76} \end{bmatrix} = \begin{bmatrix} \frac{7}{38} \\ \frac{53}{76} \\ \frac{9}{76} \end{bmatrix}$$

K Order Markov Process:

Def:

Random Process mà State hiện tại chỉ dựa vào K States trước đó

Markov Process:

Def:

Random Process có Markov Property

State Space có thể là Discrete hoặc Continuous

Markov Chain:

Def:

Markov Process với Discrete State Space

Recurrent State:

Def:

State chắc chắn sẽ trở lại khi Random Walk vô hạn lần

Reducible Markov Chain:

Def:

Markov Chain có State không được đi tới bởi State nào

Hidden Markov Model:

Def:

Kết hợp Hidden Markov Chain – Markov Chain biết trước Transition Matrix nhưng không biết Sample, và Observable State – State ta Sample được

Ex:

Cho tỉ lệ ngày nắng sau nắng là 60%, nắng sau mưa là 70%

Transition Matrix $P = \begin{bmatrix} 0.6 & 0.7 \\ 0.4 & 0.3 \end{bmatrix}$

Stationary State $X = \begin{bmatrix} \frac{7}{11} \\ \frac{4}{11} \end{bmatrix}$

Cho tỉ lệ vui khi nắng là 80%, vui khi mưa là 40%

Emission Matrix $E = \begin{bmatrix} 0.8 & 0.4 \\ 0.2 & 0.6 \end{bmatrix}$

Chúng ta không ở thành phố của thằng quan sát nên không biết nắng mưa như nào nhưng có thể liên lạc để biết nó vui hay buồn

Bài toán đặt ra là hãy dự đoán thời tiết với tỉ lệ xảy ra cao nhất

Discounted Rewards: Tránh việc sum rewards infinity

Formula: sigma(t)$y^t R(s_t, a_t)$, y E [0, 1]

MDP(Markov Decision Process): Markov Process có thêm action và reward

Actor Critic: 2 network, 1 net actor đưa ra action, 1 net critic đánh giá định hướng

Algo: tính td_error = reward + y * critic_new_state – critic_old_state

critic_loss = td_error ** 2

actor_loss = -log(actor_old_state[action]) * td_error

loss = critic_loss + actor_loss

Model: shared_layers (2 linear), actor (2 linear + softmax), critic (2 linear)

A3C(Asynchronous Advantage Actor Critic): Train nhiều agent một lúc

Algo: chạy multi process nhiều agent, update định kì sau vài step, mỗi lần update thì

sau đó cập nhật global model vào local model

CAP(Credit Assignment Problem): Khó để biết action này có tốt cho final result hay không

GAE(General Advantage Estimator): Thay cummulative rewards bằng advantages

Algo: delta(t) = r(t) + y * V(t + 1) – V(t)

A(t) = sigma(n = 0 => k – 1 – k steps) delta(t + n) * (y * λ) ** n

Bayes' theorem: Tỉ lệ giả thuyết khi cho trước một điều kiện (liên tưởng diện tích hcn)

Formula: P(H-Hypothesis|E-Event) = P(H)P(E|H)/P(E)

Ex: điều kiện E: là người mọt sách, giả thuyết H: là chủ thư viện,

có 40 nông dân với 10% mọt sách, 10 chủ thư viện với 30 % mọt sách

P(H) = 20%, P(E|H) = 30%, P(E) = 14%

Tỉ lệ người mọt sách là chủ thư viện = 3 / 7 ~ 42%

Newton's Method – Newton Raphson Method:

Def:

Dùng để tìm nghiệm gần đúng của hệ phương trình phi tuyến

Algorithm:

Bắt đầu tại một điểm bất kì

$\overrightarrow{x_0}$

Tìm hệ phương trình tuyến tính gần đúng tại điểm hiện tại bằng Jacobian

Matrix, hệ phương trình này phải đảm bảo đi qua điểm hiện tại

$J\vec{x} + f(\overrightarrow{x_0}) - J\overrightarrow{x_0} = \vec{0}$

Giải hệ phương trình này ta được tọa độ điểm mới

$J\vec{x} = J\overrightarrow{x_0} - f(\overrightarrow{x_0}) \Leftrightarrow \vec{x} = \overrightarrow{x_0} - J^{-1}f(\overrightarrow{x_0})$

Trở về bước thứ hai và lặp đi lặp lại cho đến khi hội tụ

Casio:

Giải hệ phương trình

$f_1(x, y) = 0$

$f_2(x, y) = 0$

Init

$X \leftarrow x_0$

$Y \leftarrow y_0$

Loop

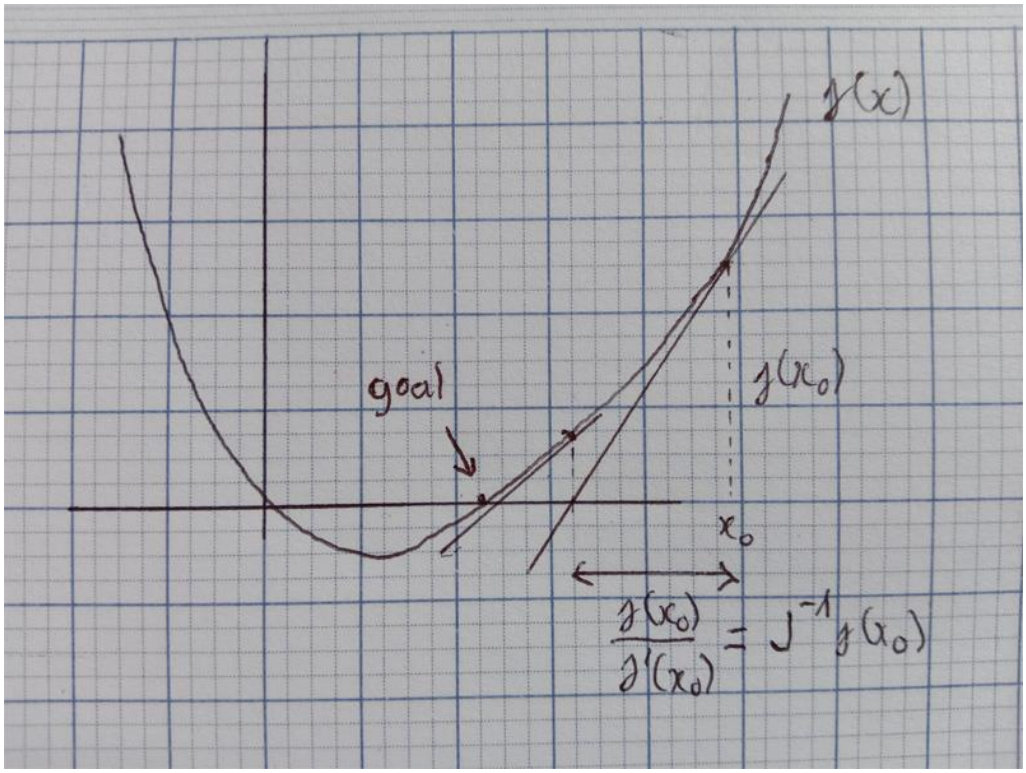
$MatA \leftarrow \begin{bmatrix} X \\ Y \end{bmatrix}$

$MatB \leftarrow \begin{bmatrix} \frac{\partial f_1}{\partial x}(X, Y) & \frac{\partial f_1}{\partial y}(X, Y) \\ \frac{\partial f_2}{\partial x}(X, Y) & \frac{\partial f_2}{\partial y}(X, Y) \end{bmatrix}$

$MatC \leftarrow \begin{bmatrix} f_1(X, Y) \\ f_2(X, Y) \end{bmatrix}$

$\begin{bmatrix} X \\ Y \end{bmatrix} \leftarrow MatA - MatB^{-1}MatC$

Intuation:



Cũ:

Search bậc 2 (superior hơn ∇ descent)

Algo: lấy đạo hàm = 0 của quadratic approximation

Formula: $x = x – \text{Hessian}(c)^{-1} @ \nabla(c)$

Conjugate Gradient Method: Giải linear system Ax = B iteratively, tránh inverse matrix, tối đa n = len(B) bước để tìm ra x, A phải là matrix đối xứng

Algo: Hệ tọa độ với các basis (d0, d1, … dn-1) thỏa mãn Q-Orthogonale – $d^T_i A d_j = 0$,

$F = 1/2x^TAx – B^Tx + C$

Init $r_0 = d_0 = -\text{First Gradient Step} = B – Ax_0$

$x_{k+1} = x_k + a_k d_k$,

$F'(x_{k+1}) = F'(x_k + a_k d_k) = d^T_k (A(x_k + a_k d_k) – B) = 0$

$\Rightarrow a_k = d^T_k(B - Ax_k) / d^T_k A d_k = d^T_k r_k / d^T_k A d_k \Rightarrow d^T_k(r_k - a_k A d_k) = 0 \Rightarrow d^T_k r_{k+1} = 0$

$\Rightarrow d^T_k r_k = (r_k + b_{k-1} d_{k-1})^T r_k = r^T r_k \Rightarrow a_k = r^T_k r_k / d^T_k A d_k = r^T_k r_k / (d_k - b_{k-1} d_{k-1})^T A d_k$

$= r^T_k r_k / r^T_k A d_k \Rightarrow (r_k - a_k A d_k)^T r_k = 0 \Rightarrow r_{k+1}{}^T r_k = 0$

$r_{k+1} = B – Ax_{k+1} = B – A(x_k + a_k d_k) = (B – Ax_k) - a_k A d_k = r_k - a_k A d_k$

$d_{k+1} = r_{k+1} + b_k d_k$

$d^T_k A d_{k+1} = 0 \Rightarrow b_k = -(r^T_{k+1} A d_k) / (d^T_k A d_k) = -(r^T_{k+1}(r_k - r_{k+1})/a_k) / (d^T_k A d_k)$

$= (r^T_{k+1} r_{k+1}) / (a_k d^T_k A d_k) = (r^T_{k+1} r_{k+1}) / (a_k r^T_k (r_k - r_{k+1})/a_k) = (r^T_{k+1} r_{k+1}) / (r^T_k r_k)$

Formula: $r_0 = d_0 = B - Ax_0$

$a_k = r^T_k r_k / d^T_k A d_k$

$x_{k+1} = x_k + a_k d_k$

$r_{k+1} = r_k - a_k A d_k$

$d_{k+1} = r_{k+1} + (r^T_{k+1} r_{k+1}) / (r^T_k r_k) * d_k$

Lagrange Multiplier: Giải bài toán optimize f(x) có constraint g(x) = k

Algo: ∇g(x) vuông góc với contour g(x), ∇f(x) vuông góc với contour g(x) nếu

∇ contour f(x) = 0 (tưởng tượng local plane)

=> ∇f(x) = a∇g(x) (2 ∇ song song, a là hệ số scale)

=> Optimize f(x) – ag(x) instead

Ex: Tìm max $f(x) = x^2 + 2y$ thỏa $g(x) = x^2 + y^2 = 1$

∇f(x) = [2x, 2], ∇g(x) = [2x , 2y], a = 1

[2x, 2] = a[2x, 2y]

$x^2 + y^2 = 1$

=>> a = 1, x = 0, y = 1, f(x) max = 2

Importance Sampling:

Def:

Sử dụng Distribution này để tính Expected Value của Distribution kia

Formula:

$$E_g[x] = \sum_x x g(x) = \sum_x x \frac{g(x)}{f(x)} f(x) = E_f\left[x \frac{g(x)}{f(x)}\right]$$

Ex:

Sử dụng Samples từ đồng xu 40% ngửa, 60% sấp để dự đoán Expected Value từ đồng xu 20% ngửa, 80% sấp, ngửa +1, sấp -1

Distribution của đồng xu 1 là f, với f(ngửa) = 40%, f(sấp) = 60%

Distribution của đồng xu 2 là g, với g(ngửa) = 20%, g(sấp) = 80%

Giả sử sau 5 lần tung đồng xu 1, kết quả là 2 lần ngửa, 3 lần sấp

$$E_g[x] = 0.2 * 1 + 0.8 * -1 = -0.6$$

$$E_g[x] = E_f\left[x\frac{g(x)}{f(x)}\right] \sim \frac{1}{5}\left(2 * 1 * \frac{0.2}{0.4} + 3 * -1 * \frac{0.8}{0.6}\right) = -0.6$$

Như vậy, ta đã dự đoán Expected Value của đồng xu 2 mà không cần tung

TRPO(Trust Region Policy Optimization): Optimize dựa vào hàm giả (giống hàm loss tại một vùng gần), dùng Actor Critic network

Algo: Tối đa loss, constraint kl <= delta, sử dụng Lagrange Multiplier

Tối đa L – b * KL, Lấy gradient của L, hessian của KL (vì 2 kia không đáng kể)

$\nabla^T(x - x_0) - b * ½ * (x - x_0)^T H(x - x_0)$

Tối ưu tại $H(x - x_0) = \nabla$, dùng Conjugate Gradient tìm $x - x_0$,

Để $KL = ½ * step(x - x_0)^T H * step(x - x_0) <= delta$,

$Step_{max} = (x - x_0) * sqrt(2 * delta / (x - x_0)^T H(x - x_0))$

Dùng linear search để kiểm tra lần nữa, giảm dần step cho đến khi loss tăng và vẫn constraint kl, step *= 0.99 ** i, i từ 0 đến n

x += step

PPO(Proximal Policy Optimization): Optimize dựa vào giới hạn step

Algo: r = tỉ lệ giữa distribution (grad) / distribution (detach)

Loss = min(r * A, clip (r, 1 – e, 1 + e) * A)

Tăng good action nhưng không > e, giảm bad action nhưng không > e

Update nhiều lần liên tiếp cho tới khi kl giữa new và old distribution > delta

A* Search Algorithm: Giống Dijkstra's Algorithm nhưng tìm kiếm kết quả không tối ưu trong thời gian ngắn

Algo: Giống Dijkstra, nhưng node hiện tại = node có giá trị value + heuristic nhỏ nhất

Heuristic = khoảng cách của node đến điểm đích (hoặc model khác)

Dừng khi tìm được đích

Hemiltonian Cycle: Loop đi qua tất cả vertices của graph một lần duy nhất

Prim's Algorithm: Thuật toán spanning tree với weight nhỏ nhất

Union-Find: Thuật toán dung hợp 2 graph cho trước 2 element thuộc 2 graph, tìm graph của

Element dựa vào node đại diện

Genetic Algorithm: Thuật toán gen di truyền dựa vào chọn lọc tiến hóa

Algo: Init đời đầu population với size = n, tạo hàm fitness đo chất lượng của cá thể

Chọn ra các cá thể với fitness lớn nhất cho thẳng vào đời sau

Tạo tiếp các cá thể tiếp theo cho đủ bằng cách hoán đổi các đoạn gen của 2 cá thể bố mẹ tốt bất kì, gây đột biến gen với tần suất thấp

Loop cho đến đời đạt yêu cầu

Steady State Selection: Chọn bố mẹ là con tốt nhất, thay thế vài con yếu bằng con của bố mẹ, phần còn lại giữ nguyên

Single Point Crossover: Chọn điểm đánh dấu, tráo gen từ điểm đó cho nhau

Ornstein-Uhlenbeck Process: Random walk process tạo brownian noise tương quan với thời gian

Formula: x_new = x_old + theta * (mean – x_old) * dt + normal(0, std) * dt ** 0.5

x_new = x_old

theta giữ cho x around mean, dt càng nhỏ càng tương đồng x_old

Xavier Initialization: Khởi tạo weight sao cho variance = 1, tránh vanishing gradient

Algo: Số neuron layer 1 * variance = 1 => variance của weight = 1/n => std scale 1/n**0.5

Random Walk: Một loạt các hành động ngẫu nhiên, 2d chắc chắn về vị trí xuất phát, 3d chưa chắc về vị trí xuất phát

Ex: Trên trục số bắt đầu tại 0, 50% sang phải 1 đơn vị, 50% sang trái 1 đơn vị, sau 2 bước, tỉ lệ về 0 là 50%, sang 2 là 25%, sang -2 là 25%, …

Brownian Motion: Chuyển động ngẫu nhiên của bụi trong fluid, là random walk với step nhỏ

Fixed Q-Network: Dùng target network trễ để tính giá trị max action

Double DQN: Chọn max action = network original, dùng target network tính value

Dueling DQN: Tách q value thành state value và advantage, sau combine V + A – mean A

DDPG(Deep Deterministic Policy Gradient): Actor Critic với Continous Action Space, lấy gradient của action

Algo: Cho 2 network, 1 Actor lấy state ra action ví dụ [1.23, 4.5], action mang giá trị thực, 1 Critic lấy state ra state value, state value concat với action ouput ra giá trị của state và action đó

Critic Loss = mse(reward + gamma * (1 - done) * target_net.critic(next state, target_net.actor(next_state), net.critic(state))

Actor Loss = -model.critic(state, model.actor(state)) => maximize giá trị q value

Model: Dùng experience replay, mới đầu explore bằng random action từ uniform distribution, sau đó random = ou_noise giảm dần, dùng target network để tính td error, ban đầu 2 network giống nhau, sau mỗi lần học, update target network bằng original network với một số lượng nhỏ, init weight = 1 / sqrt(fan_out), = 0.003 cho layer cuối

TD3(Twin Delayed DDPG): Tránh overestimate Q value

Algo: Dùng DDPG, nhưng critic sử dụng 2 network train biệt lập

Normal clip noise đơn giá trị được cộng vào target actor để tính loss

Chọn min critic trong 2 network để tính critic loss

Actor loss tính bằng critic 1, sau một vài step mới update actor

SAC(Soft Actor Critic): Dùng Stochastic policy, đồng thời maximize entropy => explore tốt

Algo: Một net actor, output mean và std, action được tính bằng mean và std

Hai net critic double q learning, Một net value giống critic và một target value update một phần mỗi lần

Khởi đầu random uniform action, tích experience replay

Sau đó sample batch, lấy actions và bonus rewards từ actor

Actions dùng reparameterization trick để có thể backprop

Bonus reward = -log(actions) | entropy + log(1 – tanh actions ** 2 + eps) | bound actions

Value(s) = Critic(s) + a * bonus reward, là tổng giá trị state và bonus từ hiện tại

Target value = min(2 critic với state và actors) + a * bonus reward, lấy loss mse

Actor loss = mean -target value

Critic loss = td error reward + gamma * target value net(next state) * (1 - done),

update 2 critic

Update target value net một phần

NAF(Normalized Advantage Function): Dùng quadratic advantage, continuous action space

Algo: Một net mean action, một net std action, một net value, một target value

Khởi đầu random uniform action, tích experience replay

Sau đó sample action từ mean và std (đã exp để positive),

Sample batch state, action, reward, new_state, done

Target value = reward + gamma * (1 - done) * target_value(new_state)

Q_value = value(state) – 0.5 * ((action – mean_action) ** 2) dot std ** -2

Value loss = mse(target_value, q_value)

Update net, target value một phần

PER – Prioritized Experience Replay:

Def:

Ưu tiên lựa chọn học những nước đi sai lầm nhất dựa vào Temporal Difference Error

Kết hợp với các Off-Policy Algorithm sử dụng Experience Replay như DQN, DDPG

Algo:

Khởi tạo Experience Replay kèm theo Priority mỗi Transition, Network, Priority Rate $\propto$, Importance Rate β (khởi đầu với giá trị nhỏ)

For 1 – Max Episode:

For 1 – Max Step:

Chọn Action, Step

Lưu State, Goal, New State, Action, Reward, Done vào Experience Replay

Lưu New Priority = Max Priorities (Khởi đầu = 1) vào Experience Replay

Sample Batch từ Experience Replay với tỉ lệ được chọn mỗi phần tử $\rho_i = \frac{p_i^{\alpha}}{\sum p^{\alpha}}$ (ρ là Probability, p là Priority)

$w_i = \left(\frac{1}{N\rho_i}\right)^{\beta}$ (w là Imporance Weight, N là độ lớn Experience Replay hiện tại)

$w_i = \frac{w_i}{Max\ w}$ (để Normalize)

Học và Update Network với Importance Weight mỗi phần tử nhân với Gradient của phần tử đó

Update Priorities ứng với Sample với Absolute Temporal Difference Error + Eps (để tránh Priority = 0)

State = New State

Tăng β

HER – Hindsight Experience Replay:

Def:

Khi môi trường cho Sparse Reward, học bằng cách thông thường rất khó, nên để tao Dense Reward, ta giả định trong quá khứ Goal không phải là Goal mong muốn mà là State cuối cùng đạt được trong Episode, ví dụ nếu như Goal là State đó thì chúng ta đã Win

Tăng khả năng Generalize trên các Goal khác nhau

Kết hợp với các Off-Policy Algorithm sử dụng Experience Replay như DQN, DDPG

Algo:

Khởi tạo Experience Replay, Network

For 1 – Max Episode:

Episode Transitions = States, New States, Actions, Rewards = []

Sample Goal mong muốn

For 1 – Max Step:

Chọn Action, Step

Lưu State, Goal, New State, Action, Reward, Done vào Experience Replay

Lưu State, New State, Action, Reward vào Episode Transitions

Sample Batch từ Experience Replay rồi học

State = New State

New Goal = Last State

For Transition trong Transitions:

Nếu là Transition cuối cùng thì Reward = Phần thưởng khi Win

Lưu State, New Goal, New State, Action, Reward, Done trong Transition vào Experience Replay

Sample Batch từ Experience Replay rồi học