HỆ THỐNG TÌM KIẾM (PHẦN 2)

Trần Trung Kiên

ttkien@fit.hcmus.edu.vn

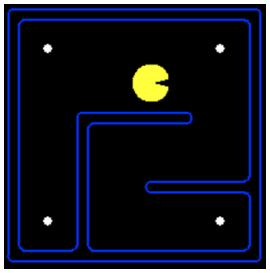
Tổng thể

- Ôn lại buổi học trước
  - Bài toán tìm kiếm
  - Các thuật toán tìm kiếm không sử dụng thông tin trạng thái đích để định hướng: DFS, BFS, UCS
- Các thuật toán tìm kiếm sử dụng thông tin trạng thái đích để định hướng

Bức tranh lớn

Bài toán

Vd, tìm đường đi cho Pacman đến góc trái dưới



Định nghĩa bài toán tìm kiếm (người làm): xác định không gian trạng thái, từ đó định nghĩa hàm successor, hàm kiểm tra trạng thái đích, trạng thái bắt đầu

Chạy thuật toán tìm kiếm (AI làm)

Lời giải: kế hoạch gồm một chuỗi các hành động để đi từ trạng thái bắt đầu đến trạng thái đích

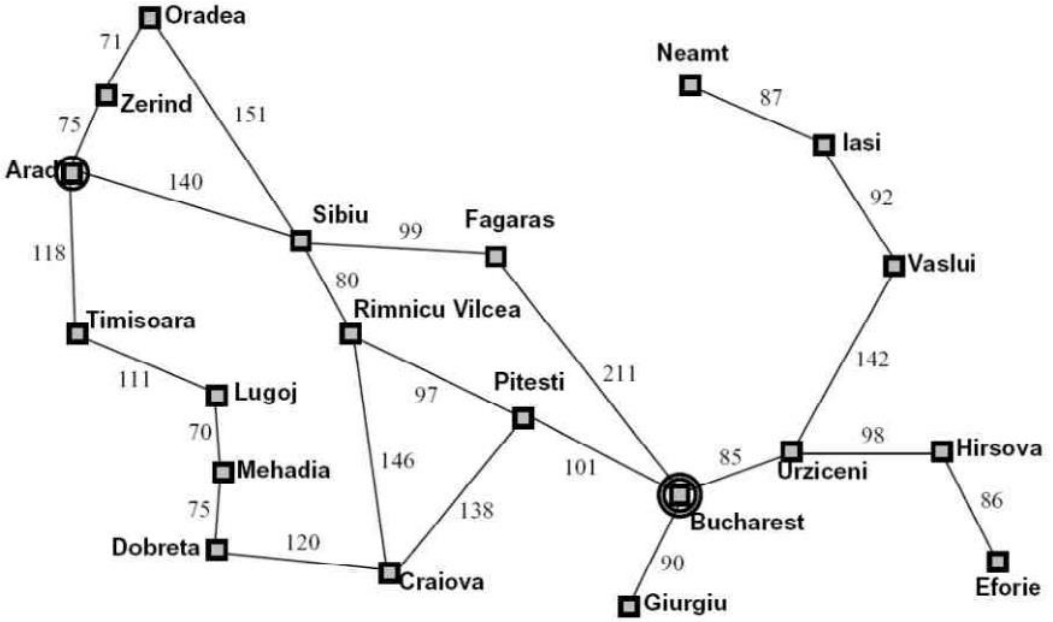
Khi bài toán thay đổi, ta sẽ cần định nghĩa lại bài toán tìm kiếm ☹ …

… nhưng thuật toán tìm kiếm sẽ không phải thay đổi ☺

Tập luyện định nghĩa bài toán tìm kiếm

Tìm đường đi từ thành phố Arad đến thành phố Bucharest (nước Rumani)

- Tập trạng thái: tập thành phố
- Hàm successor: với một thành phố, trả về các thành phố láng giềng cùng với hành động là đường đi và chi phí là chiều dài đường đi
- Trạng thái bắt đầu: Arad
- Hàm kiểm tra trạng thái đích: trạng thái có bằng Bucharest?

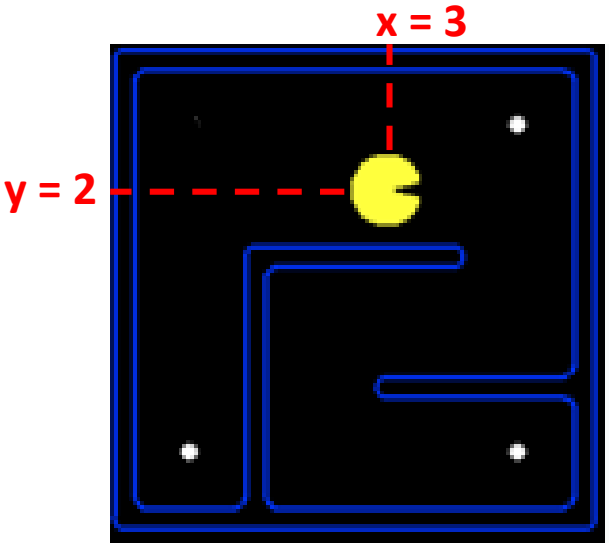


Nguồn ảnh: UC Berkely. CS188 Intro to AI

Tập luyện định nghĩa bài toán tìm kiếm

Tìm đường đi cho Pacman để đến được bốn góc của mê cung

- Tập trạng thái: tập (vị trí Paman, mảng bool cho biết góc nào đã được đi tới)
- Hàm successor: từ một trạng thái, cho biết có thể đi đến những trạng thái nào với hành động nào (Đ, T, N, B) và chi phí bao nhiêu (có thể coi tất cả các hành động đều có chi phí bằng nhau và bằng 1)
- Trạng thái bắt đầu: ((3, 2), (false, false, false, false))
- Hàm kiểm tra trạng thái đích: mảng bool có bằng true hết?



Ôn lại các thuật toán tìm kiếm

Đầu vào: trạng thái bắt đầu, hàm successor, hàm kiểm tra trạng thái đích

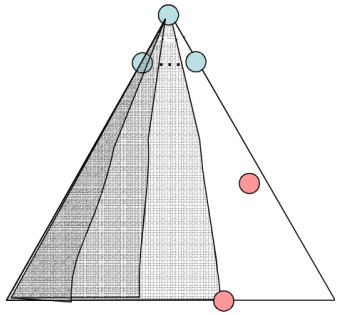
Đầu ra: kế hoạch tìm được (chuỗi các hành động để đi từ trạng thái bắt đầu đến trạng thái đích)

Quá trình thực hiện:

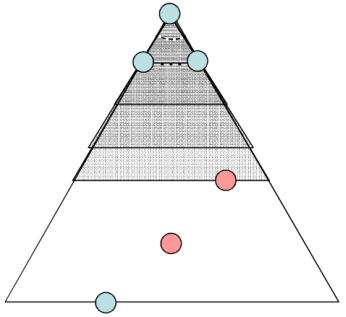
- Khởi tạo fringe: gồm một kế hoạch ứng với trạng thái bắt đầu
- Trong khi fringe chưa rỗng:
  - Lấy một kế hoạch ra khỏi fringe theo một chiến lược nào đó
  - Kiểm xem kế hoạch này có đi tới G (dùng hàm kiểm tra trạng thái đích)?
    - Nếu có → PARTY!
    - Nếu không → mở rộng ra (dựa vào hàm successor) và đưa các kế hoạch mới vào fringe

Ôn lại các thuật toán tìm kiếm

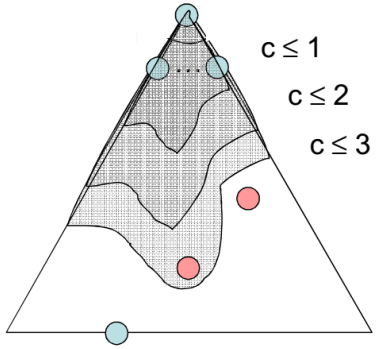
- **DFS (Depth-First Search)**
  - Chiến lược chọn kế hoạch từ fringe để mở rộng: chọn kế hoạch "sâu" nhất → Có thể dùng stack để cài đặt fringe
  - Không đảm bảo tìm được kế hoạch có chi phí thấp nhất



- **BFS (Breadth-First Search)**
  - Chiến lược chọn kế hoạch từ fringe để mở rộng: chọn kế hoạch "nông" nhất → Có thể dùng queue để cài đặt fringe
  - Không đảm bảo tìm được kế hoạch có chi phí thấp nhất

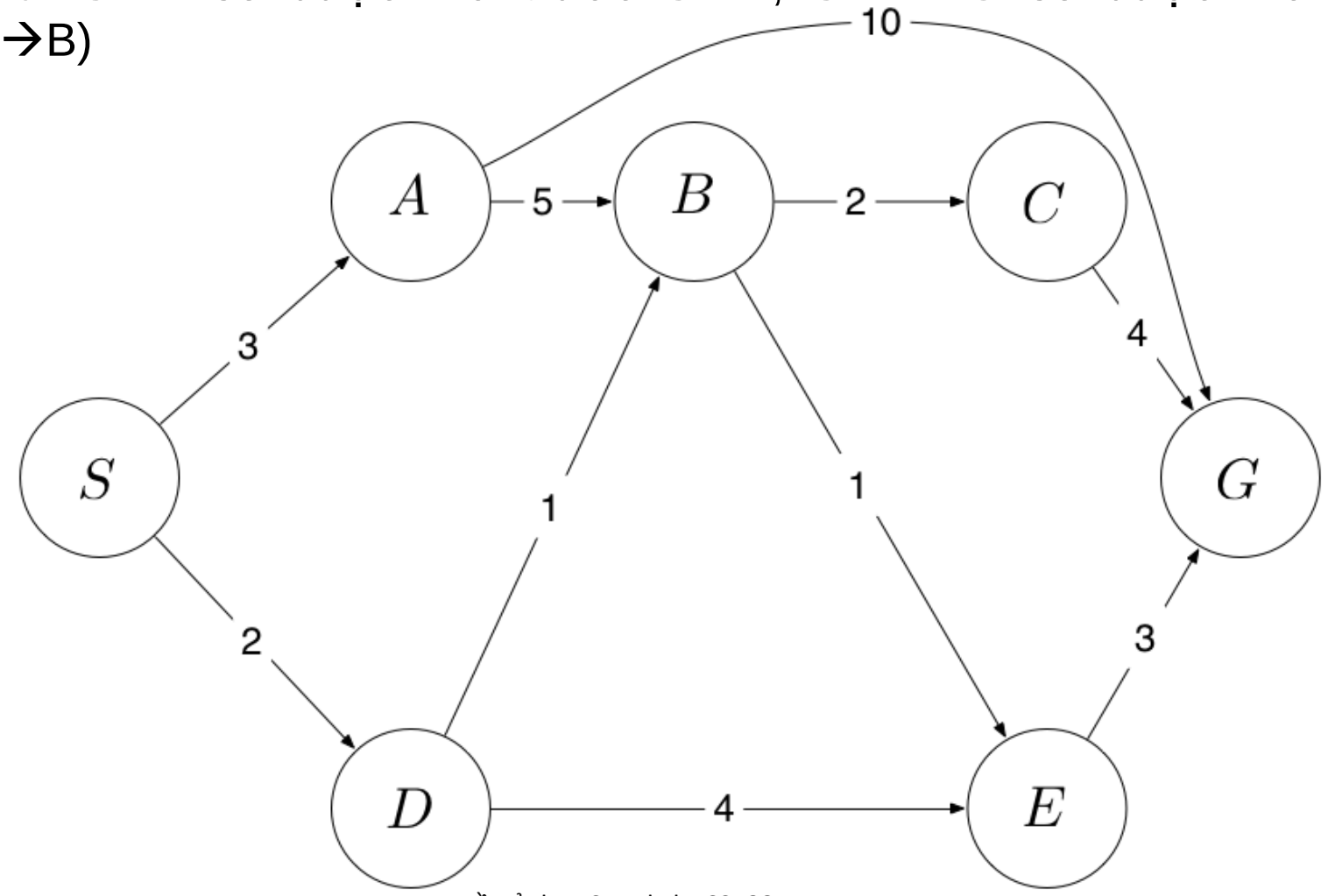


- **UCS (Uniform Cost Search)**
  - Chiến lược chọn kế hoạch từ fringe để mở rộng: chọn kế hoạch có chi phí thấp nhất → Có thể dùng priority queue để cài đặt fringe
  - Đảm bảo tìm được kế hoạch có chi phí thấp nhất



Chạy DFS với đồ thị ở dưới

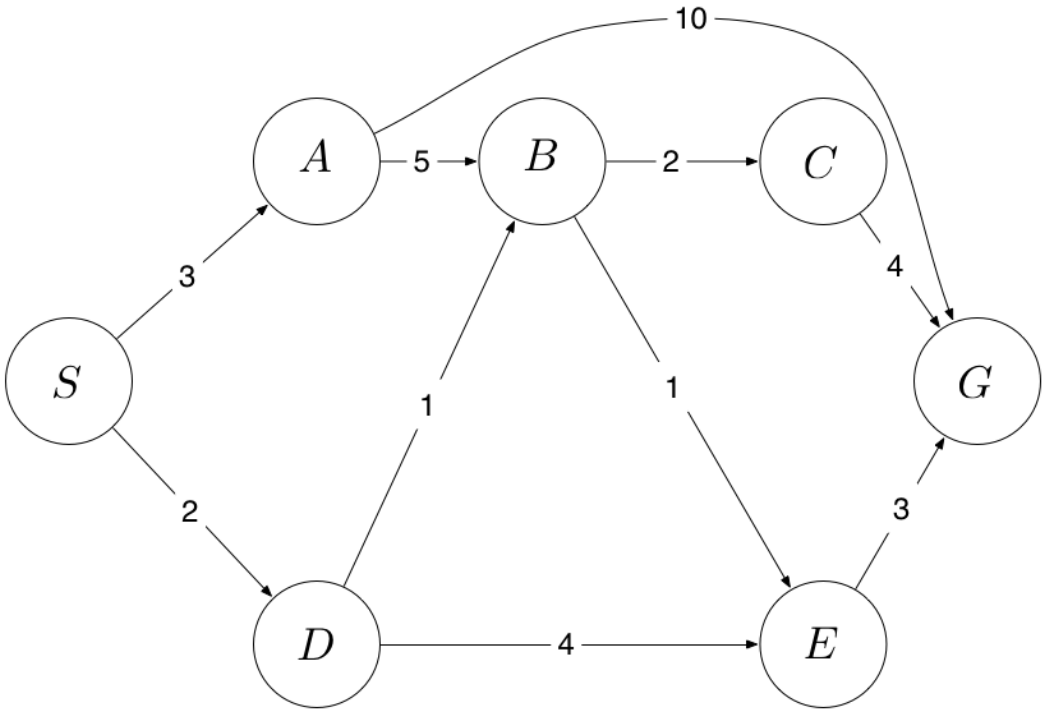
Giả sử nếu các kế hoạch có cùng độ sâu thì ưu tiên theo thứ tự bảng chữ cái (vd: S→A sẽ được mở trước S→B, S→A→G sẽ được mở trước S→D→B)



Nguồn ảnh: UC Berkely. CS188 Intro to AI

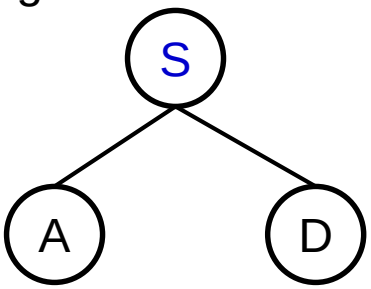
Khởi tạo fringe

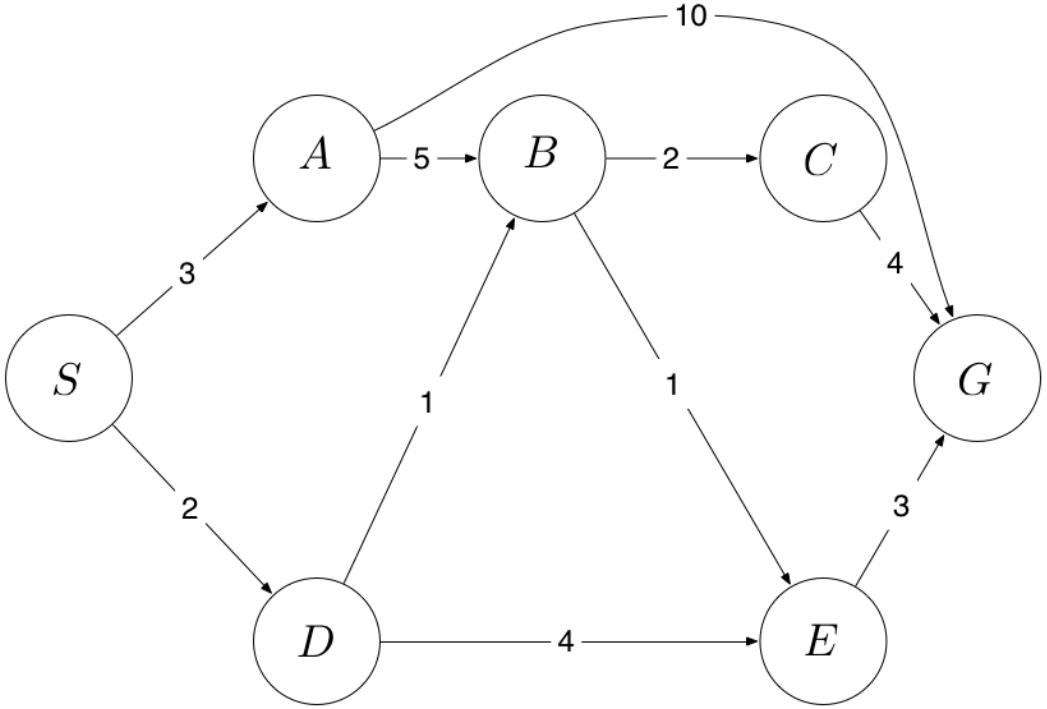
S



Trong khi fringe chưa rỗng (lần 1):

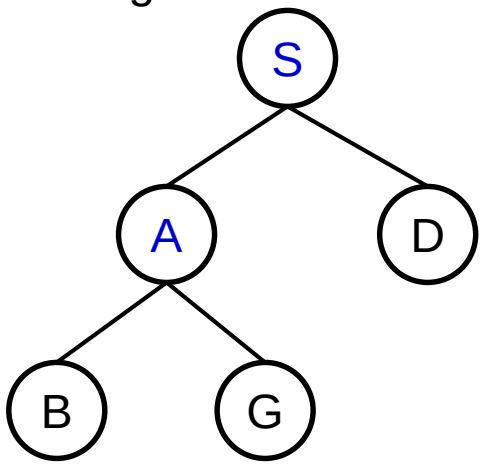
- Lấy kế hoạch "sâu" nhất ra khỏi fringe
- Kiểm xem kế hoạch này có đi tới G?
  - Nếu có → PARTY!
  - Nếu không → mở rộng ra và đưa các kế hoạch mới vào fringe

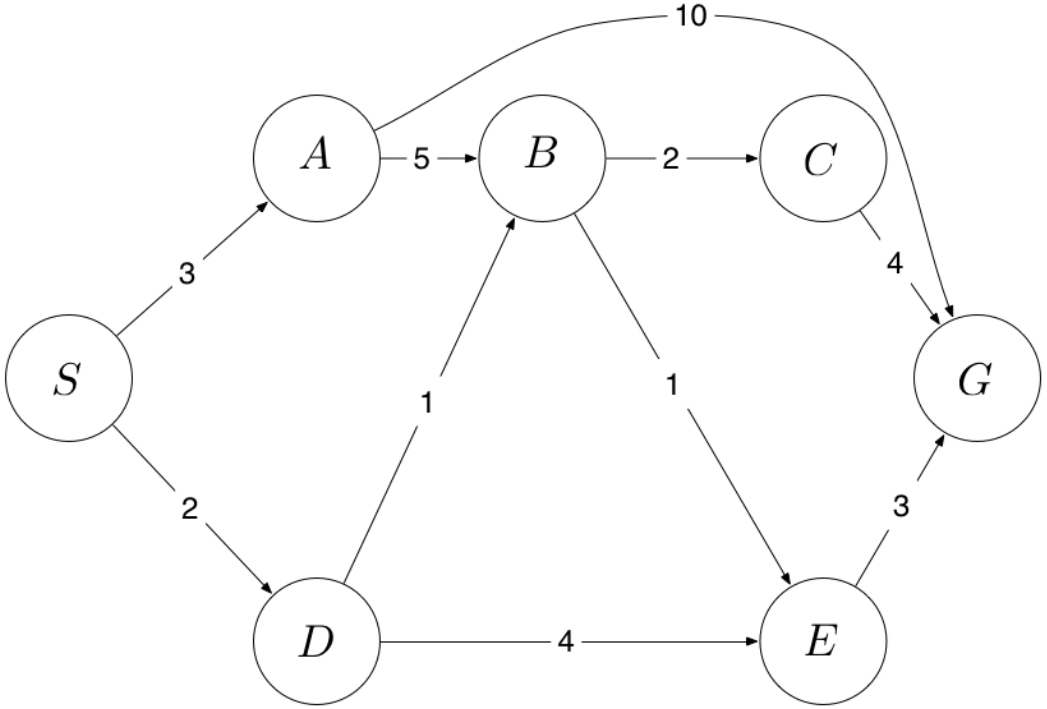




Trong khi fringe chưa rỗng (lần 2):

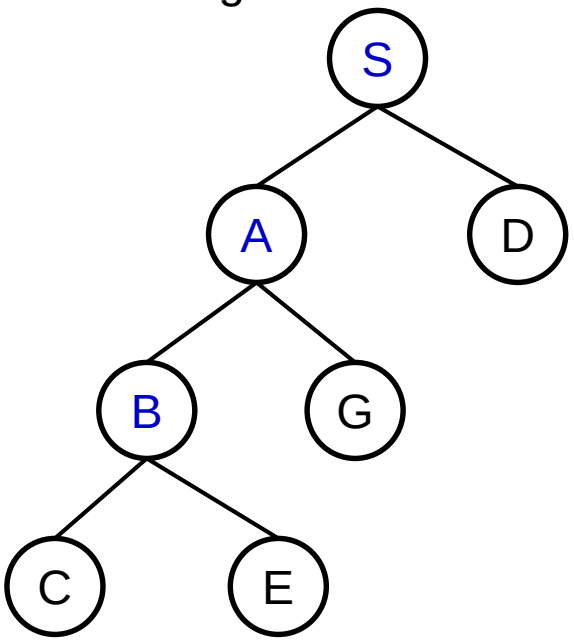
- Lấy kế hoạch "sâu" nhất ra khỏi fringe
- Kiểm xem kế hoạch này có đi tới G?
  - Nếu có → PARTY!
  - Nếu không → mở rộng ra và đưa các kế hoạch mới vào fringe

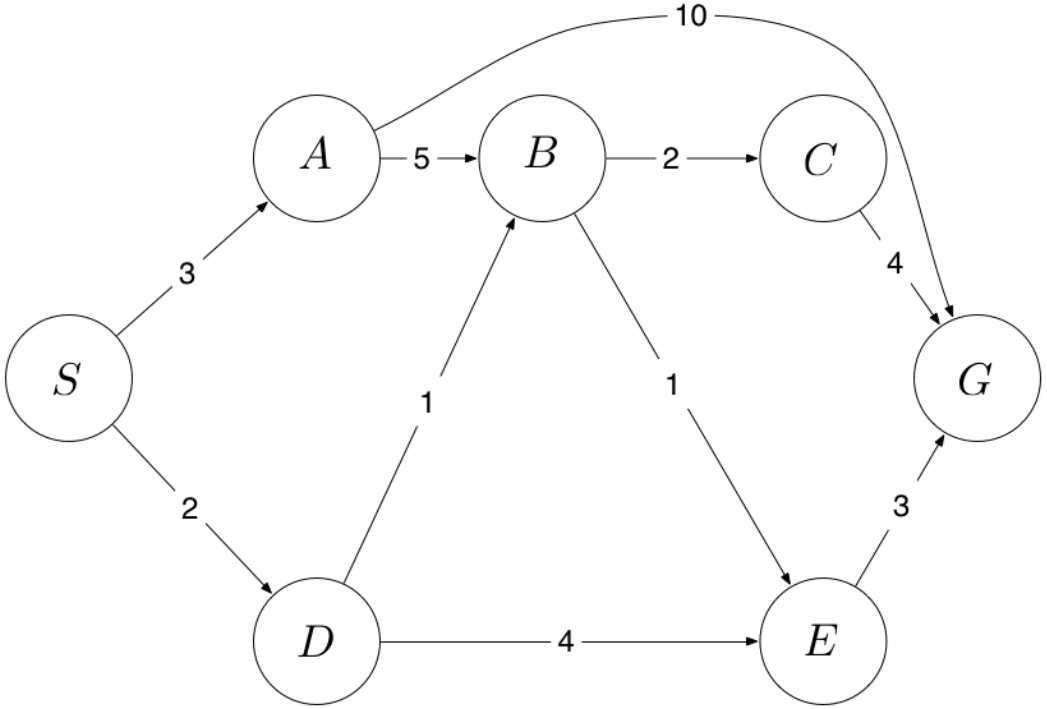




Trong khi fringe chưa rỗng (lần 3):

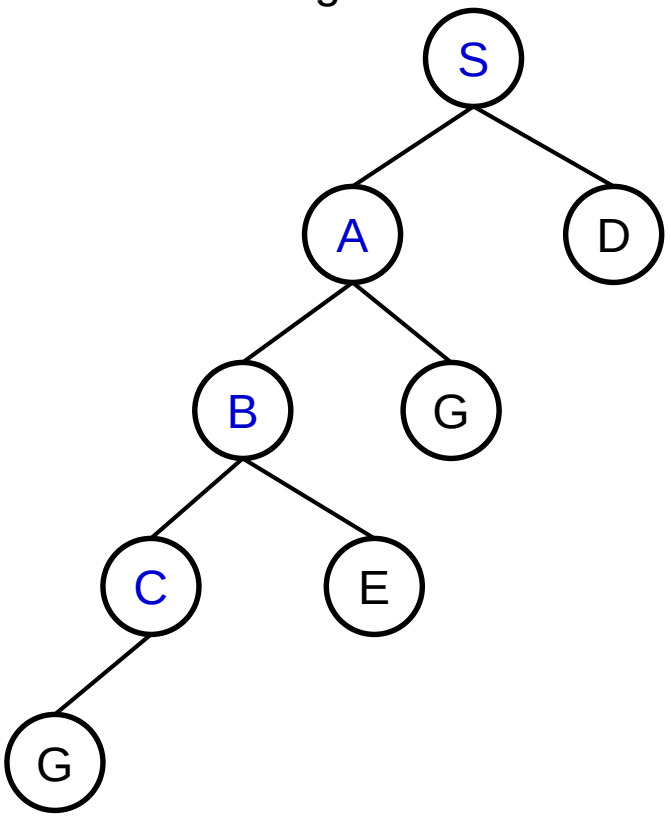
- Lấy kế hoạch "sâu" nhất ra khỏi fringe
- Kiểm xem kế hoạch này có đi tới G?
  - Nếu có → PARTY!
  - Nếu không → mở rộng ra và đưa các kế hoạch mới vào fringe

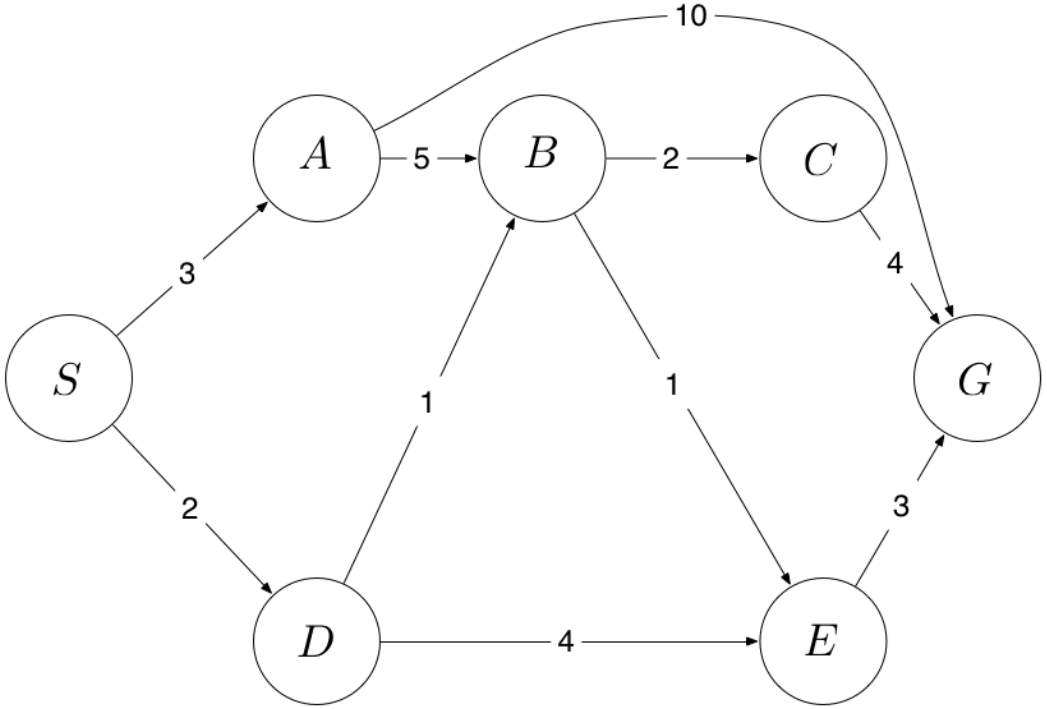




Trong khi fringe chưa rỗng (lần 4):

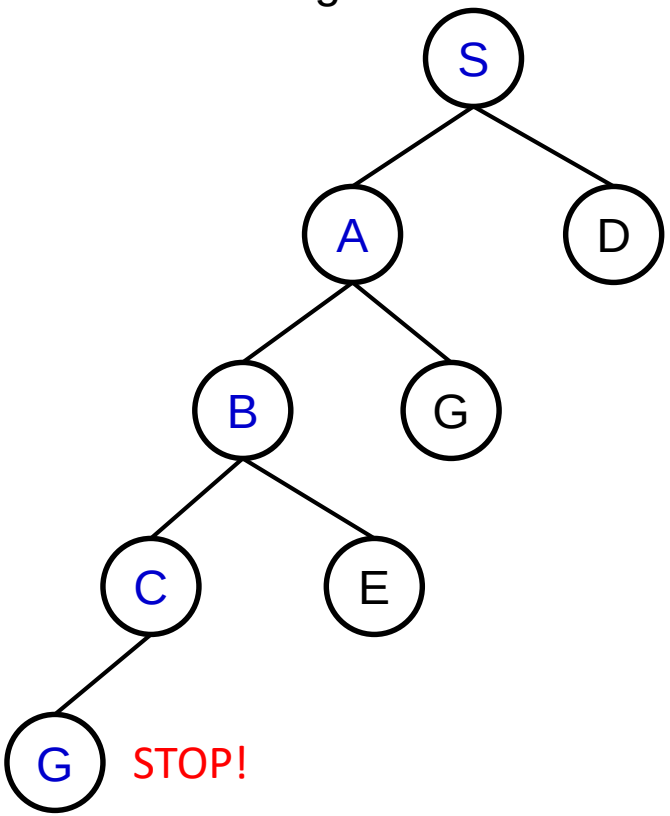
- Lấy kế hoạch "sâu" nhất ra khỏi fringe
- Kiểm xem kế hoạch này có đi tới G?
  - Nếu có → PARTY!
  - Nếu không → mở rộng ra và đưa các kế hoạch mới vào fringe

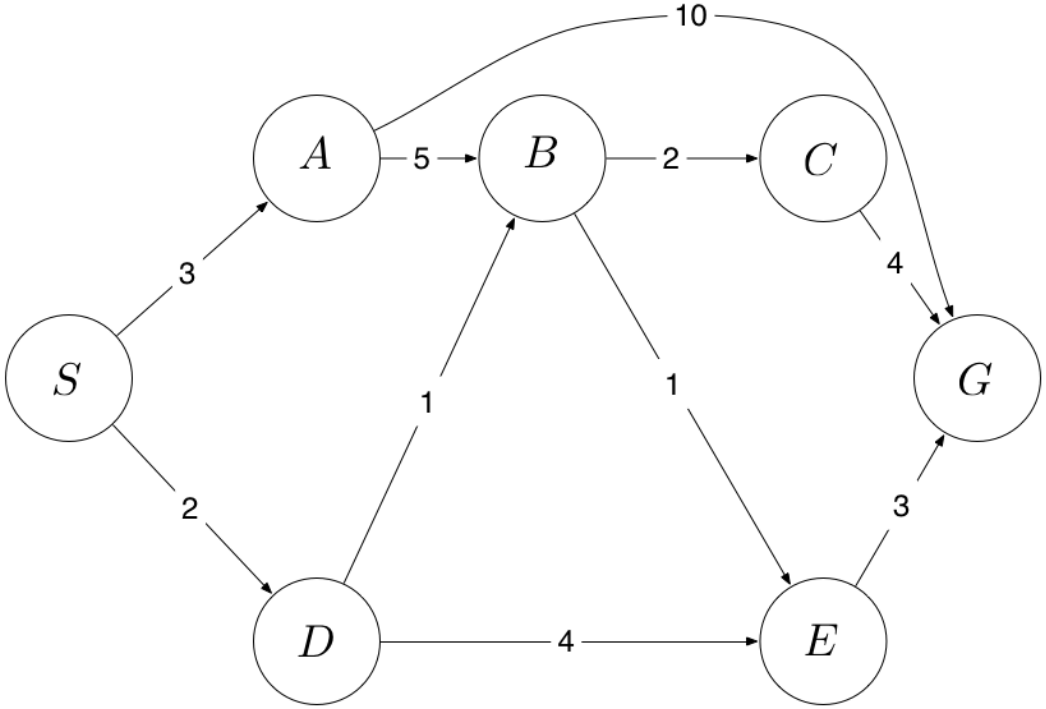




Trong khi fringe chưa rỗng (lần 5):

- Lấy kế hoạch "sâu" nhất ra khỏi fringe
- Kiểm xem kế hoạch này có đi tới G?
  - Nếu có → PARTY!
  - Nếu không → mở rộng ra và đưa các kế hoạch mới vào fringe



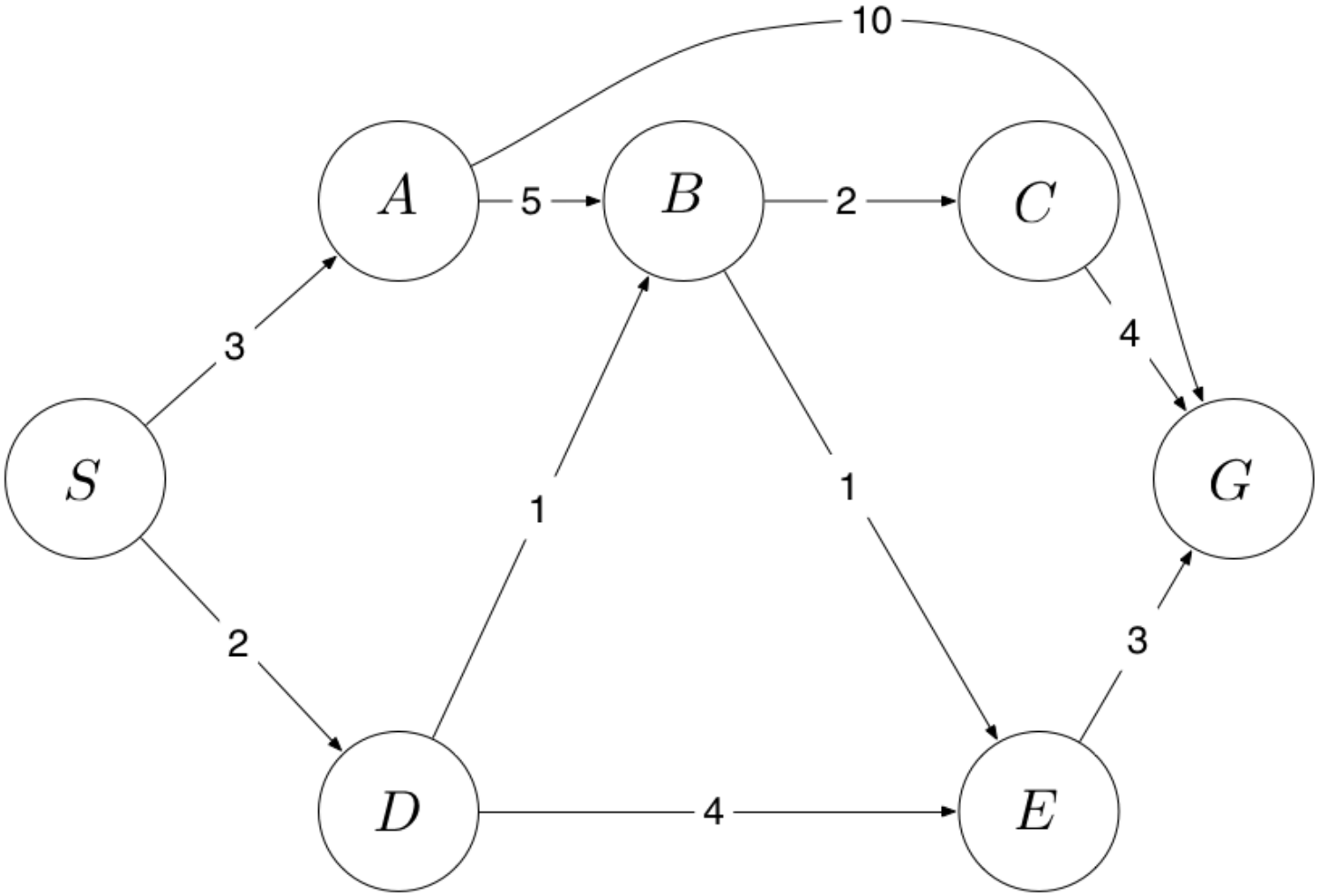


Dùng stack để cài đặt fringe của DFS

- Stack sẽ thể hiện được tư tưởng chọn kế hoạch "sâu" nhất của DFS
- Nguyên lý hoạt động của stack là "vào sau ra trước" → các kế hoạch được đưa vào stack sau cùng sẽ được lấy ra trước, mà các kế hoạch được đưa vào stack sau cùng là các kế hoạch "sâu" nhất
  - Ví dụ với đồ thị vừa làm …

Chạy BFS với đồ thị ở dưới

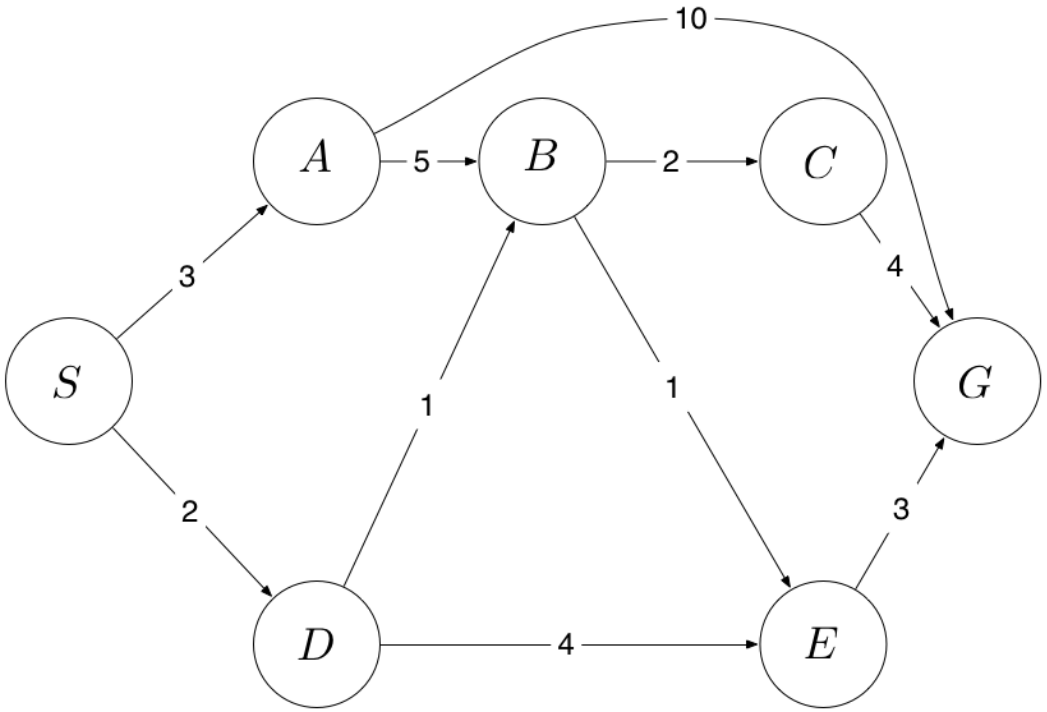
Giả sử nếu các kế hoạch có cùng độ sâu thì ưu tiên theo thứ tự bảng chữ cái



Nguồn ảnh: UC Berkely. CS188 Intro to AI

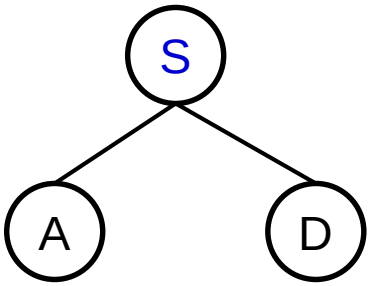
Khởi tạo fringe

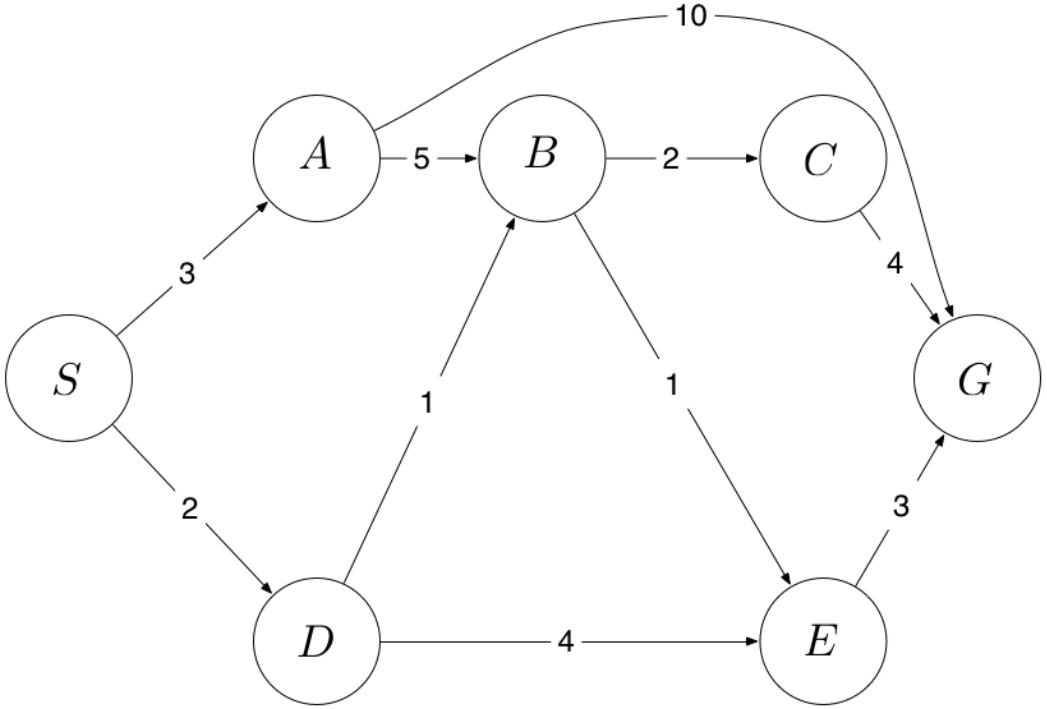
S



Trong khi fringe chưa rỗng (lần 1):

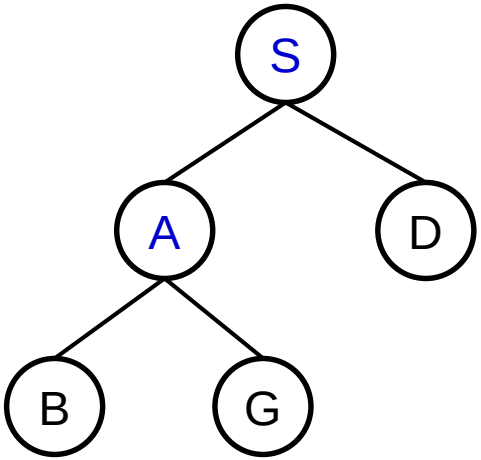
- Lấy kế hoạch "nông" nhất ra khỏi fringe
- Kiểm xem kế hoạch này có đi tới G?
  - Nếu có → PARTY!
  - Nếu không → mở rộng ra và đưa các kế hoạch mới vào fringe

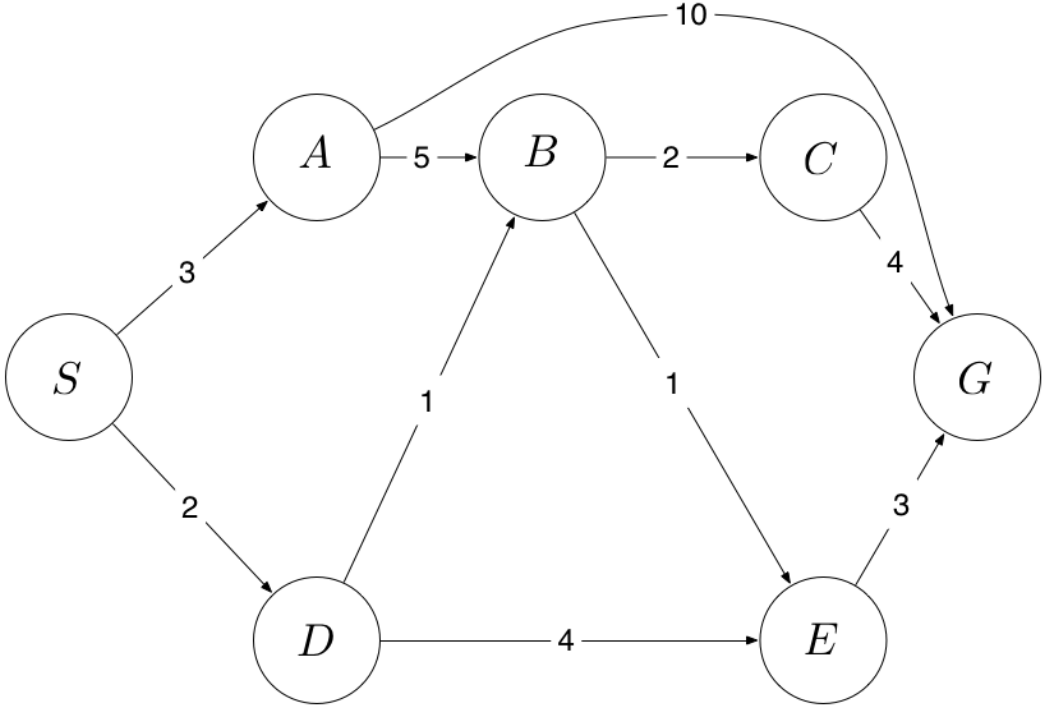




Trong khi fringe chưa rỗng (lần 2):

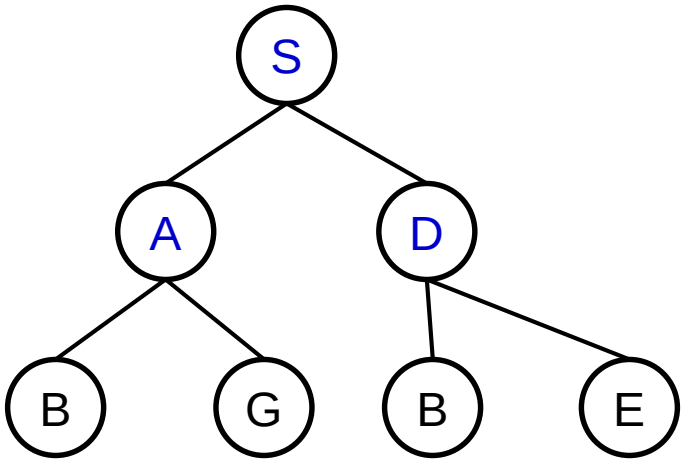
- Lấy kế hoạch "nông" nhất ra khỏi fringe
- Kiểm xem kế hoạch này có đi tới G?
  - Nếu có → PARTY!
  - Nếu không → mở rộng ra và đưa các kế hoạch mới vào fringe

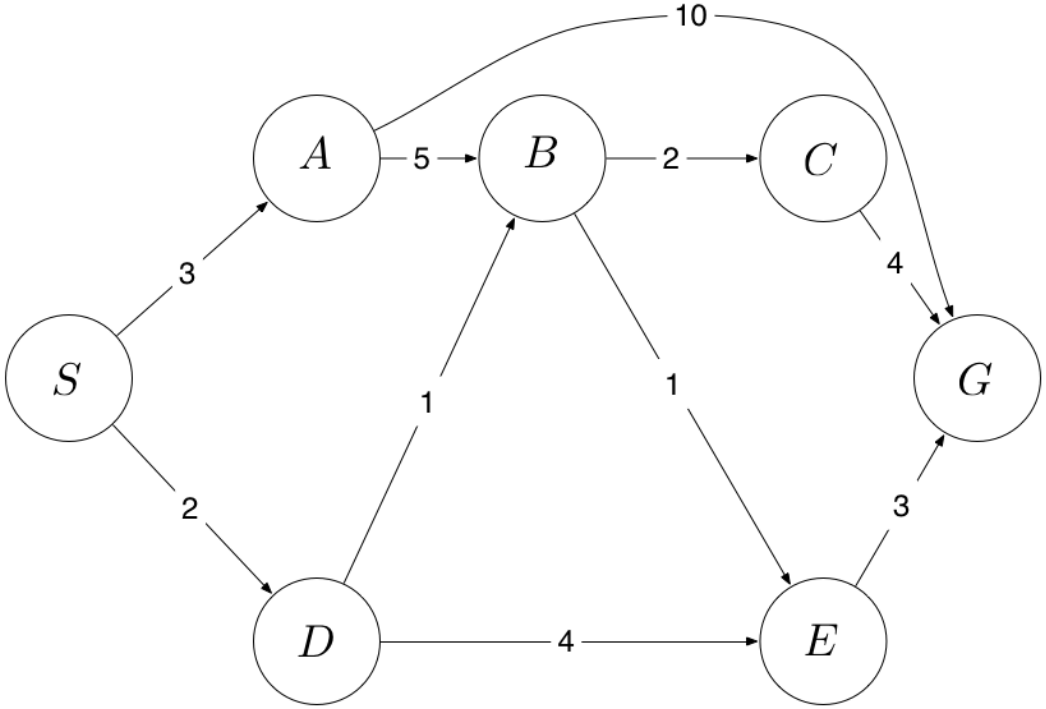




Trong khi fringe chưa rỗng (lần 3):

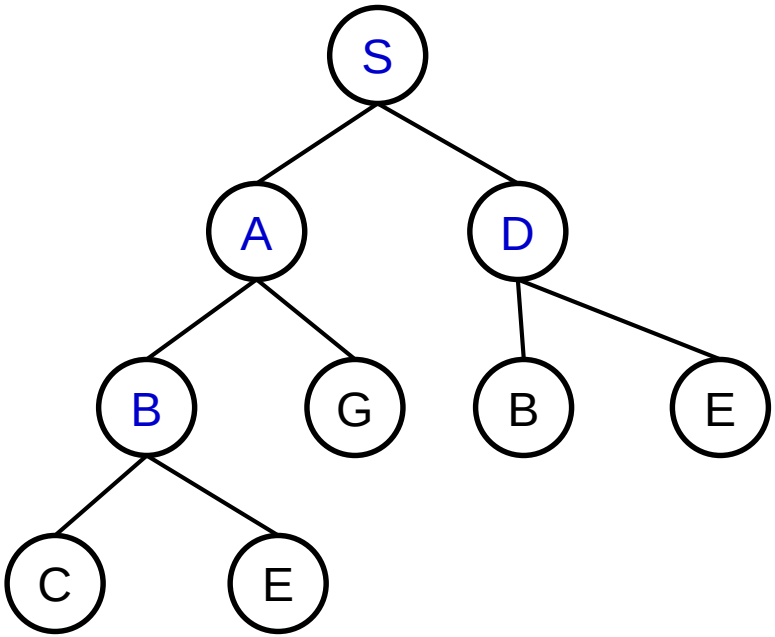
- Lấy kế hoạch "nông" nhất ra khỏi fringe
- Kiểm xem kế hoạch này có đi tới G?
  - Nếu có → PARTY!
  - Nếu không → mở rộng ra và đưa các kế hoạch mới vào fringe

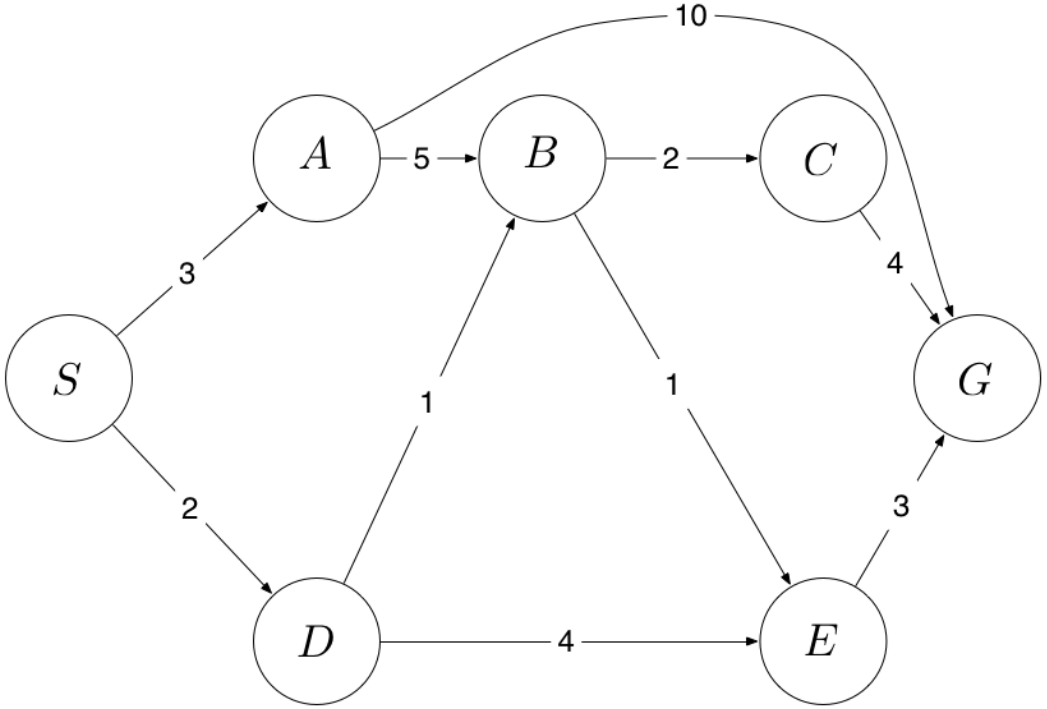




Trong khi fringe chưa rỗng (lần 4):

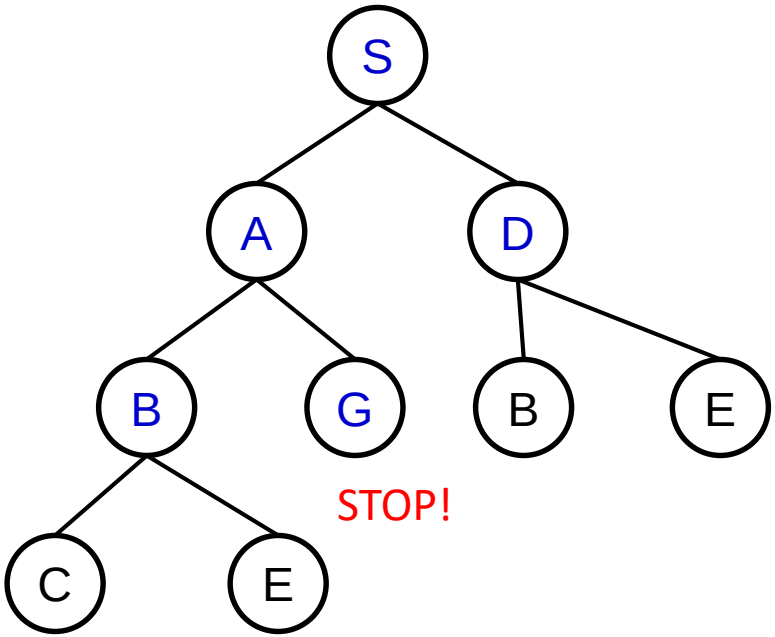
- Lấy kế hoạch "nông" nhất ra khỏi fringe
- Kiểm xem kế hoạch này có đi tới G?
  - Nếu có → PARTY!
  - Nếu không → mở rộng ra và đưa các kế hoạch mới vào fringe

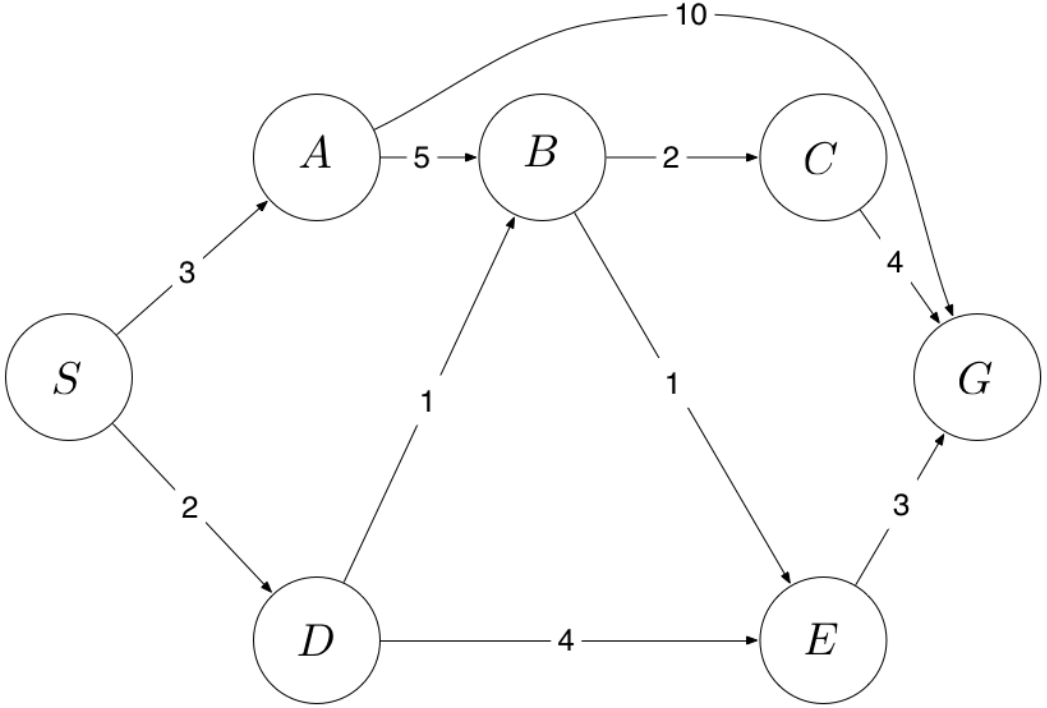




Trong khi fringe chưa rỗng (lần 5):

- Lấy kế hoạch "nông" nhất ra khỏi fringe
- Kiểm xem kế hoạch này có đi tới G?
  - Nếu có → PARTY!
  - Nếu không → mở rộng ra và đưa các kế hoạch mới vào fringe



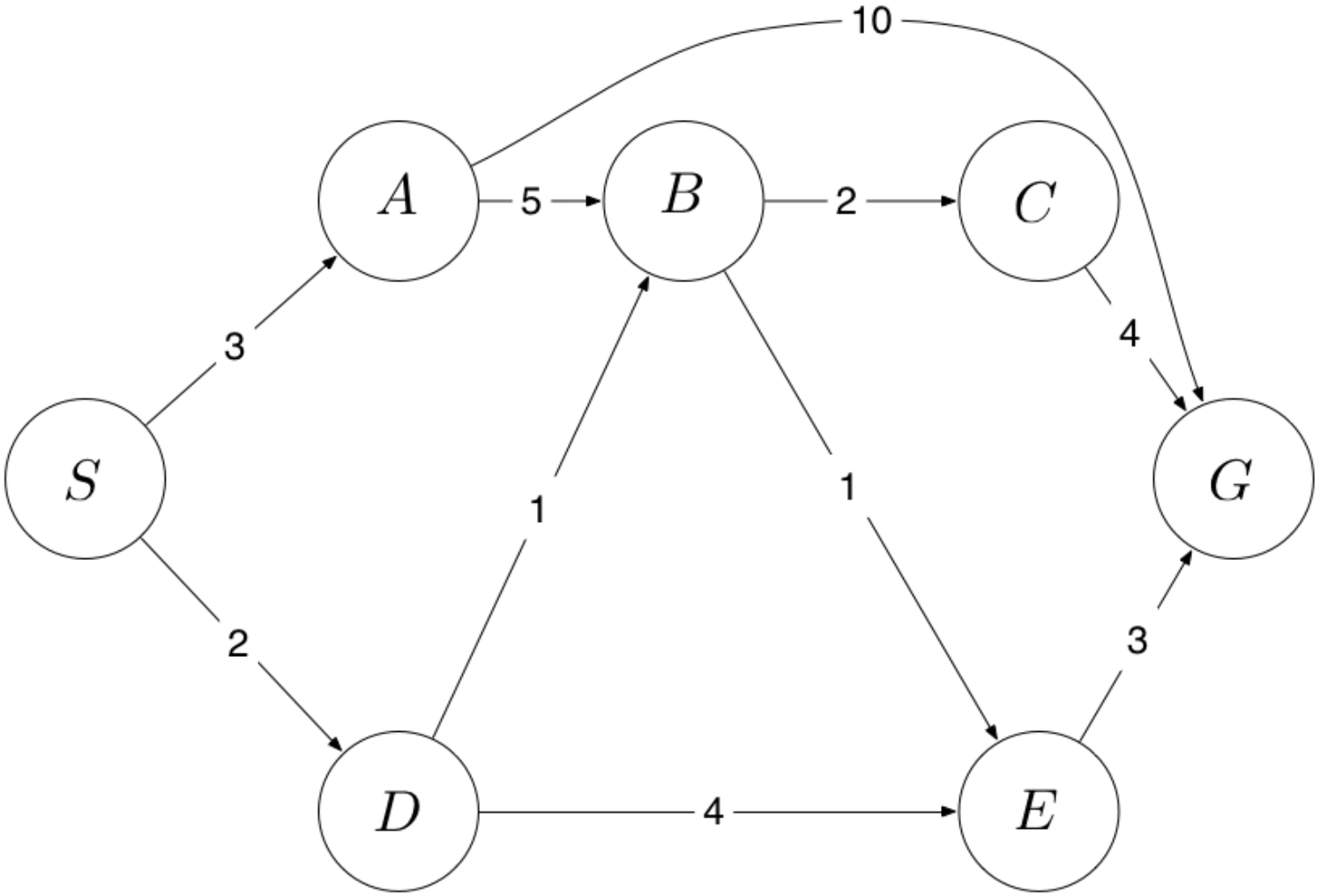


Dùng queue để cài đặt fringe của BFS

- Queue sẽ thể hiện được tư tưởng chọn kế hoạch "nông" nhất của BFS
- Nguyên lý hoạt động của queue là "vào trước ra trước" → các kế hoạch được đưa vào stack trước tiên sẽ được lấy ra trước, mà các kế hoạch được đưa vào stack trước tiên là các kế hoạch "nông" nhất
  - Ví dụ với đồ thị vừa làm …

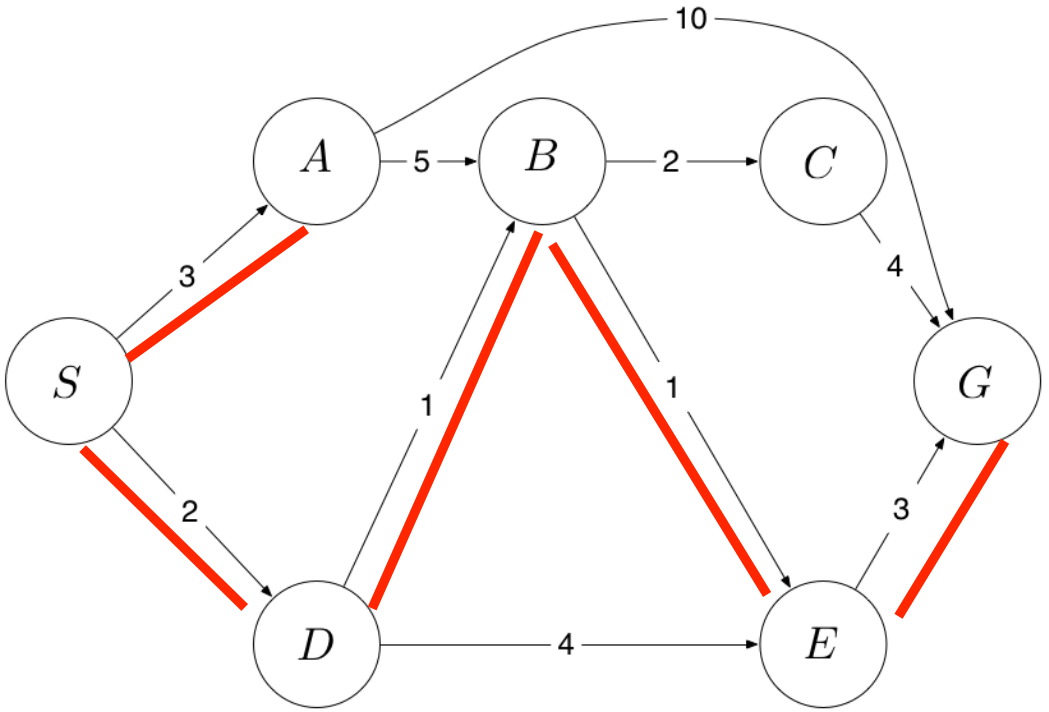
Chạy UCS với đồ thị ở dưới

Giả sử nếu các kế hoạch có cùng chi phí thì ưu tiên theo thứ tự bảng chữ cái



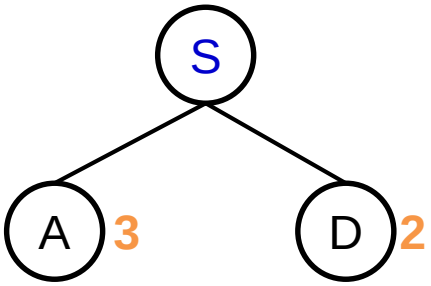
Nguồn ảnh: UC Berkely. CS188 Intro to AI

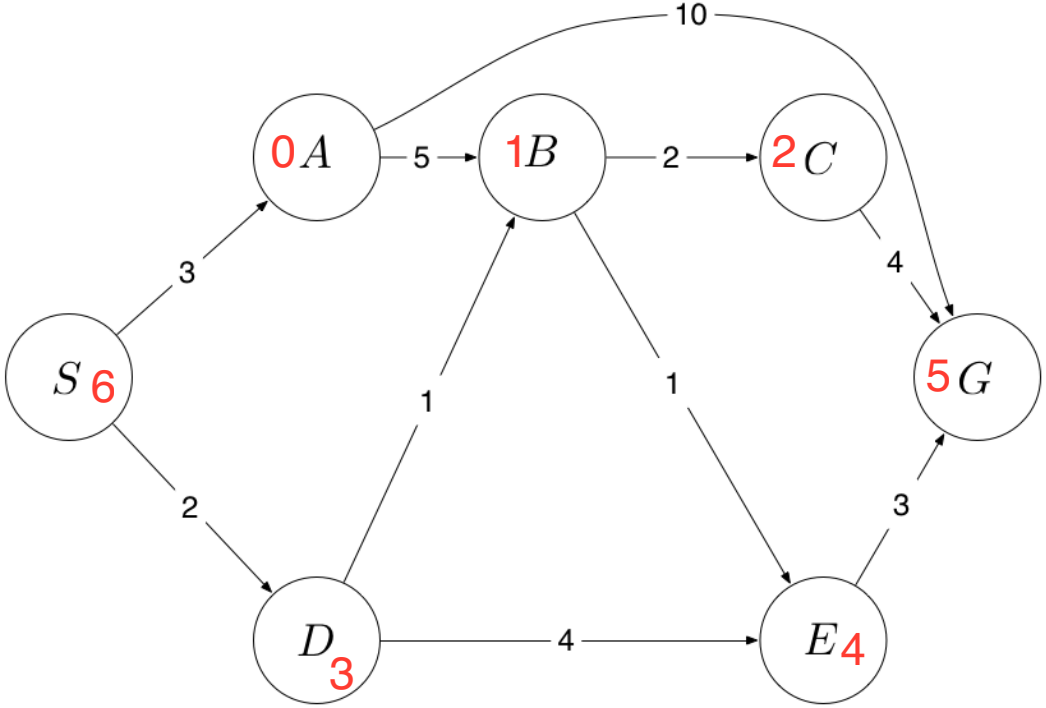
Khởi tạo fringe



Trong khi fringe chưa rỗng (lần 1):

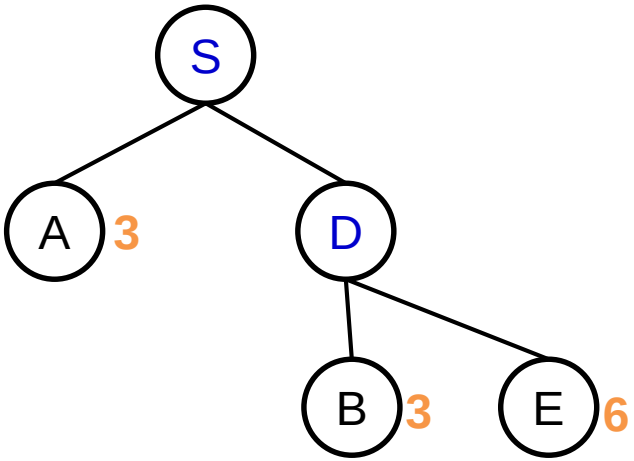
- Lấy kế hoạch có chi phí thấp nhất ra khỏi fringe
- Kiểm xem kế hoạch này có đi tới G?
  - Nếu có → PARTY!
  - Nếu không → mở rộng ra và đưa các kế hoạch mới vào fringe

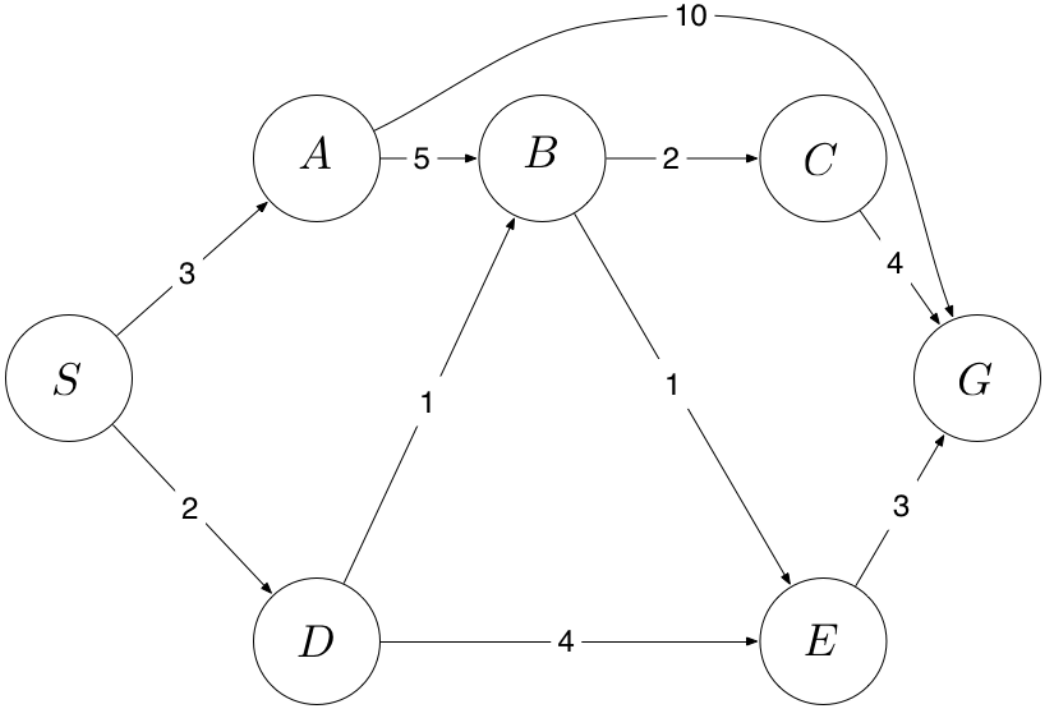




Trong khi fringe chưa rỗng (lần 2):

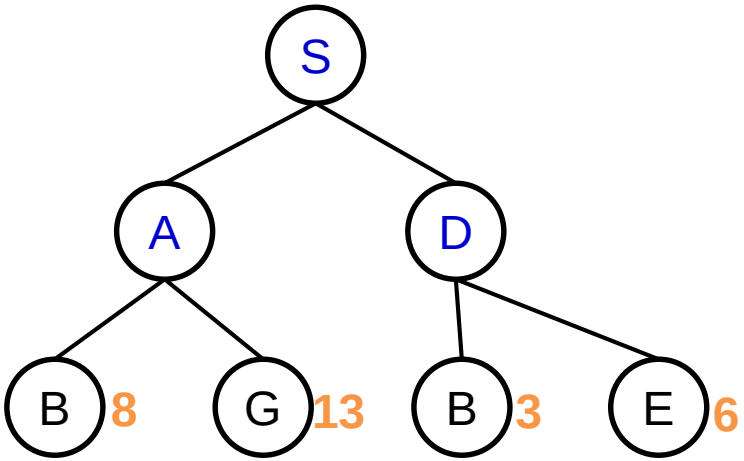
- Lấy kế hoạch có chi phí thấp nhất ra khỏi fringe
- Kiểm xem kế hoạch này có đi tới G?
  - Nếu có → PARTY!
  - Nếu không → mở rộng ra và đưa các kế hoạch mới vào fringe

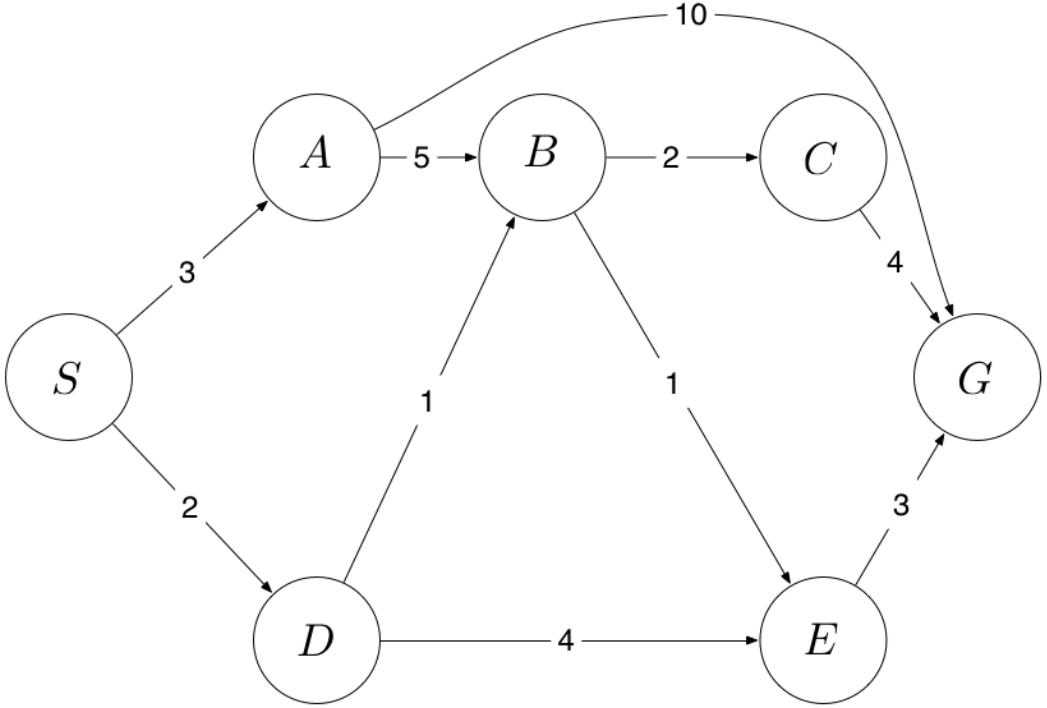




Trong khi fringe chưa rỗng (lần 3):

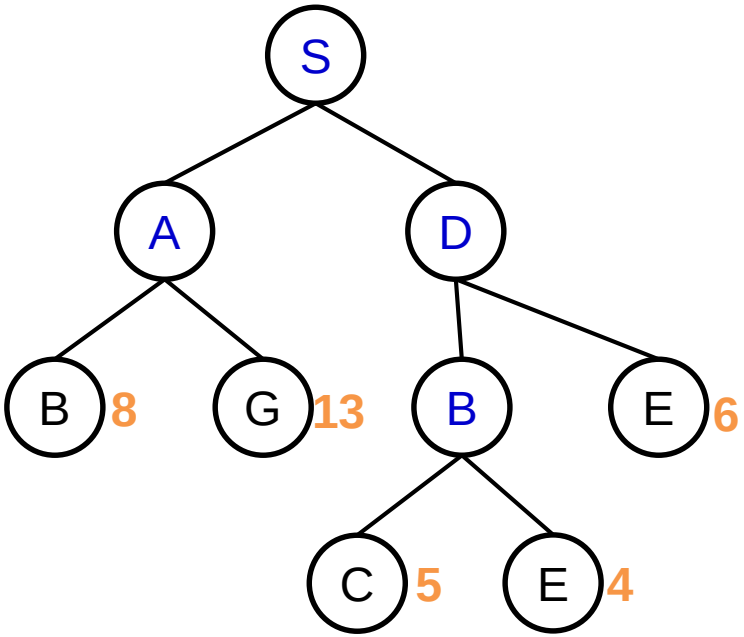
- Lấy kế hoạch có chi phí thấp nhất ra khỏi fringe
- Kiểm xem kế hoạch này có đi tới G?
  - Nếu có → PARTY!
  - Nếu không → mở rộng ra và đưa các kế hoạch mới vào fringe

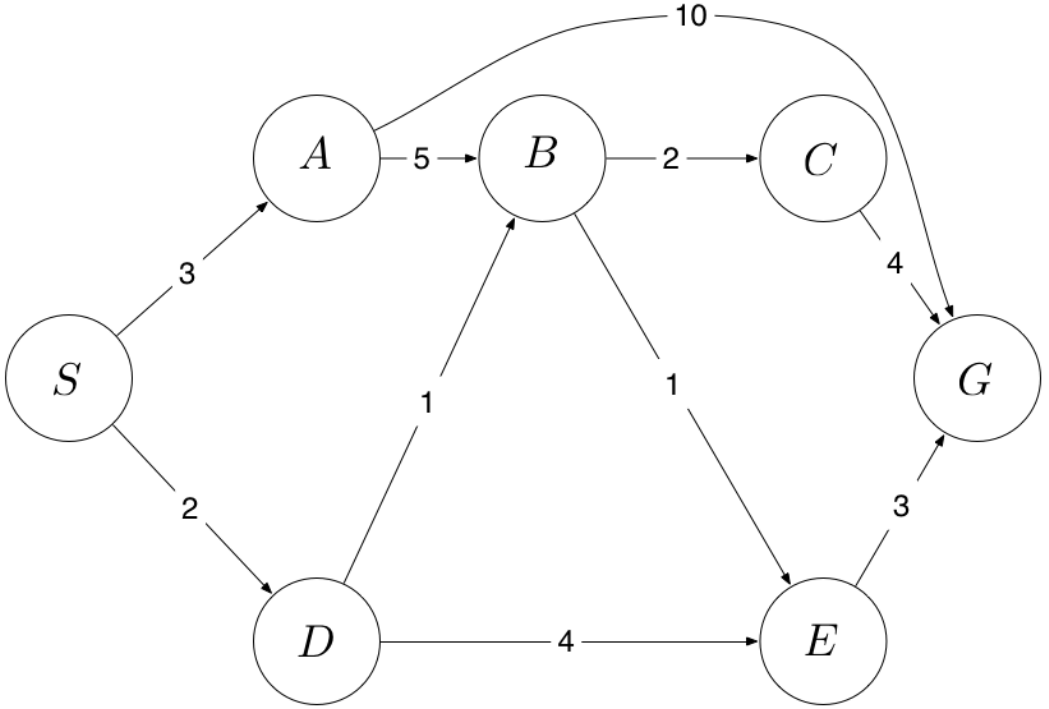




Trong khi fringe chưa rỗng (lần 4):

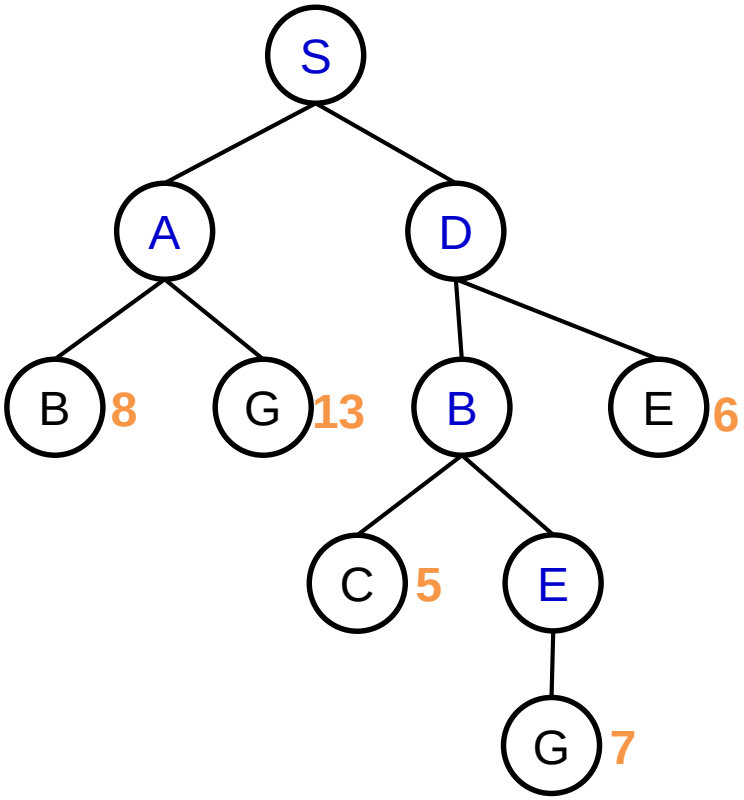
- Lấy kế hoạch có chi phí thấp nhất ra khỏi fringe
- Kiểm xem kế hoạch này có đi tới G?
  - Nếu có → PARTY!
  - Nếu không → mở rộng ra và đưa các kế hoạch mới vào fringe

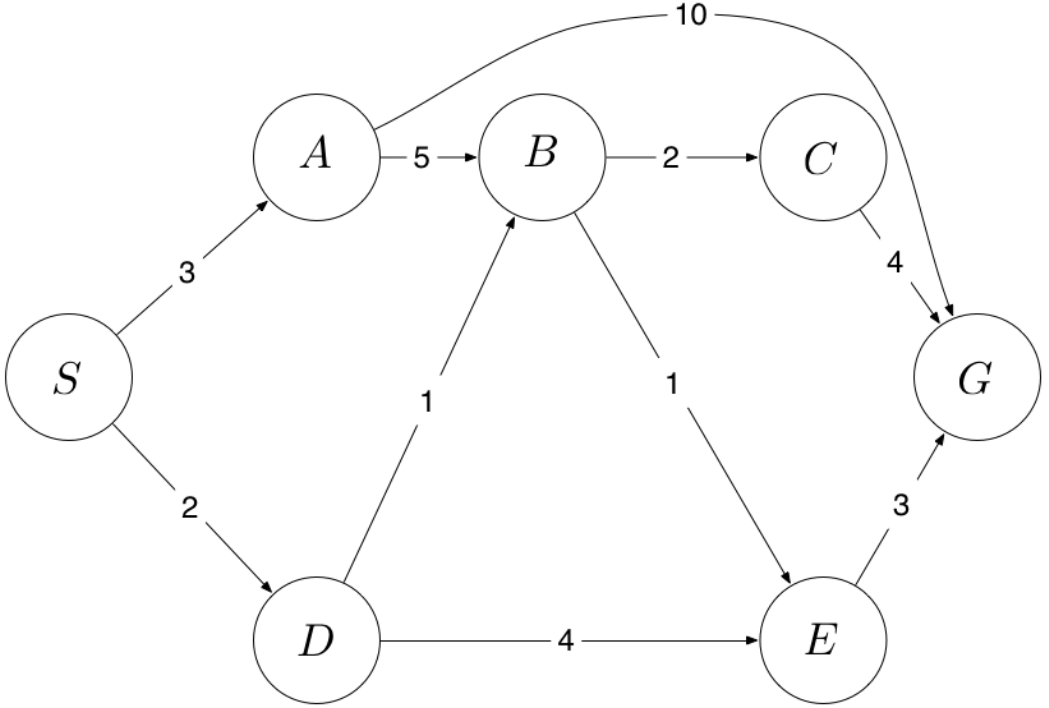




Trong khi fringe chưa rỗng (lần 5):

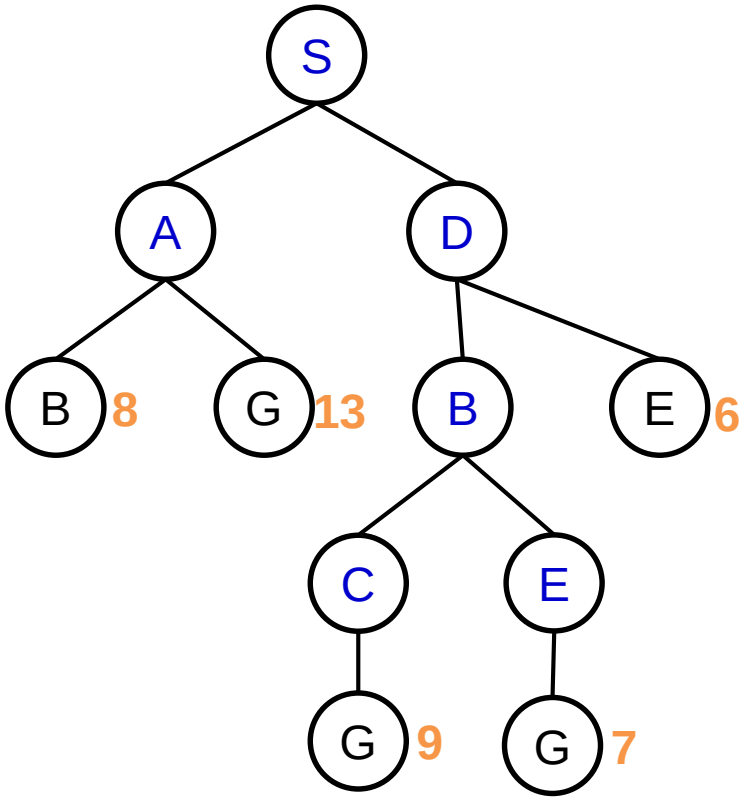
- Lấy kế hoạch có chi phí thấp nhất ra khỏi fringe
- Kiểm xem kế hoạch này có đi tới G?
  - Nếu có → PARTY!
  - Nếu không → mở rộng ra và đưa các kế hoạch mới vào fringe

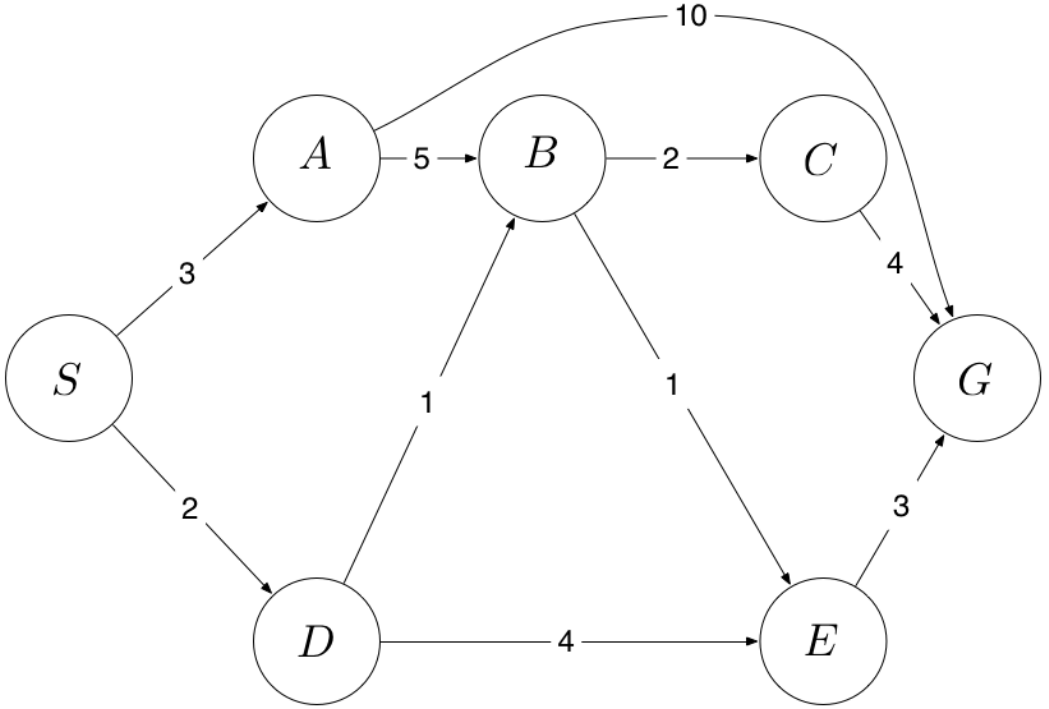




Trong khi fringe chưa rỗng (lần 6):

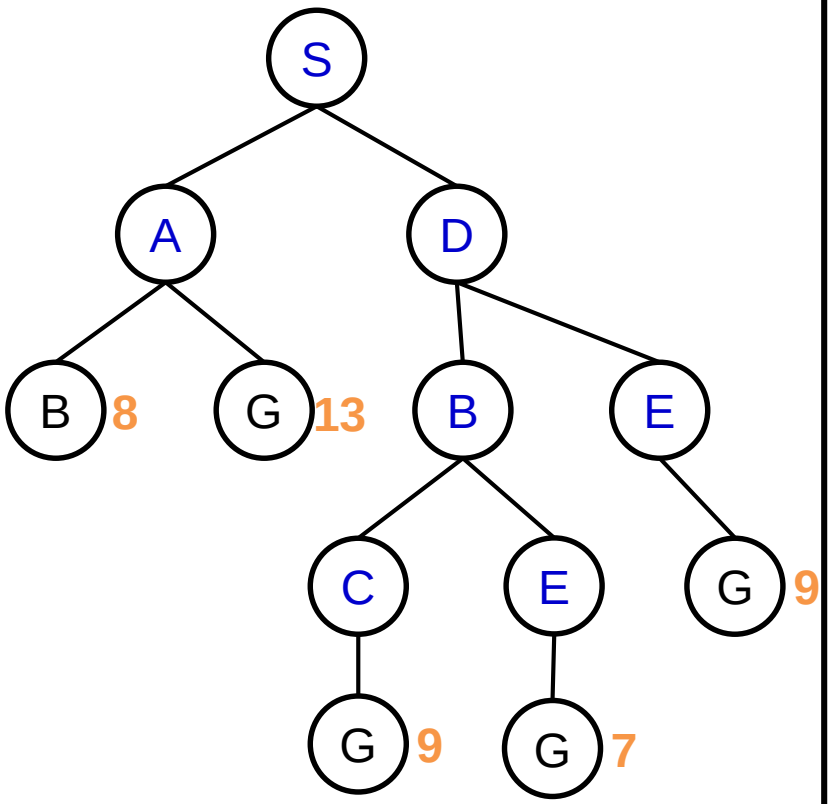
- Lấy kế hoạch có chi phí thấp nhất ra khỏi fringe
- Kiểm xem kế hoạch này có đi tới G?
  - Nếu có → PARTY!
  - Nếu không → mở rộng ra và đưa các kế hoạch mới vào fringe

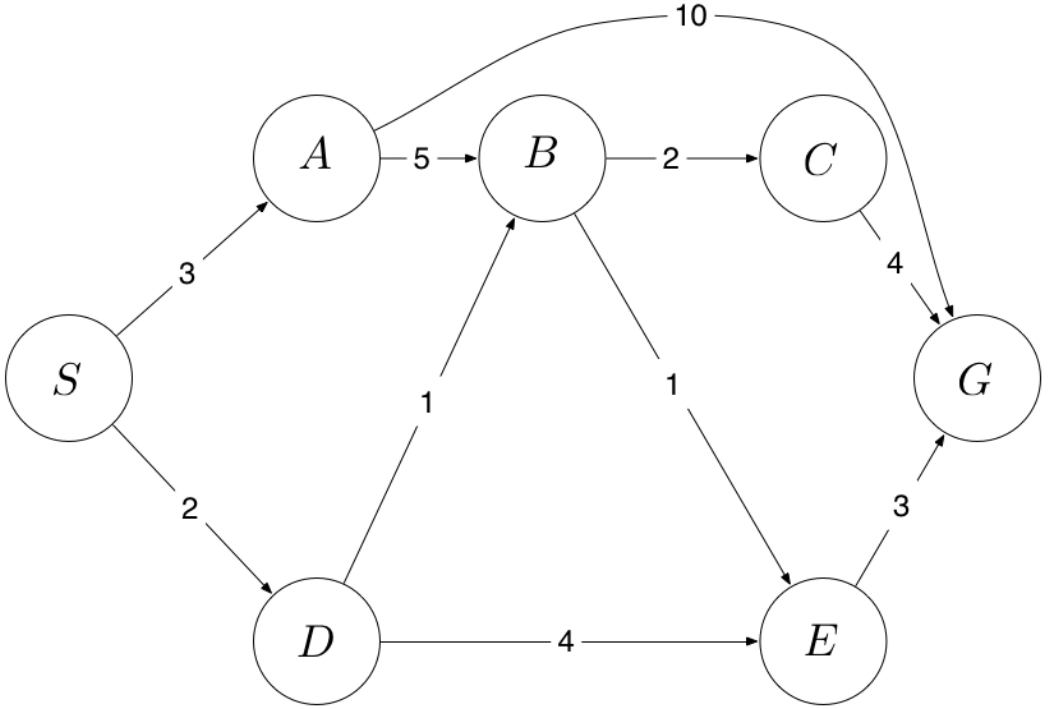




Trong khi fringe chưa rỗng (lần 7):

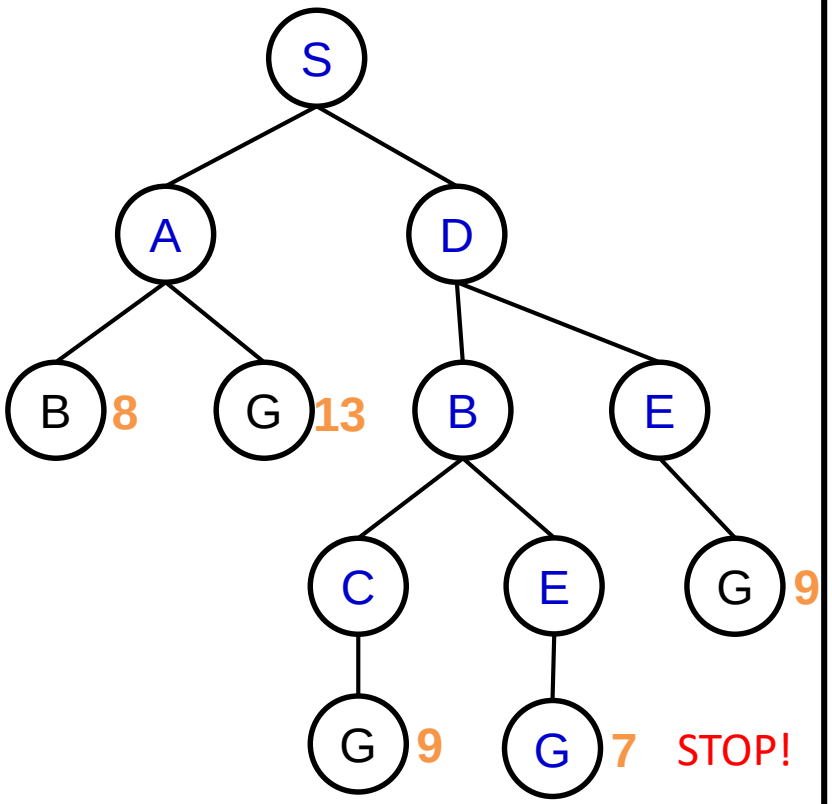
- Lấy kế hoạch có chi phí thấp nhất ra khỏi fringe
- Kiểm xem kế hoạch này có đi tới G?
  - Nếu có → PARTY!
  - Nếu không → mở rộng ra và đưa các kế hoạch mới vào fringe

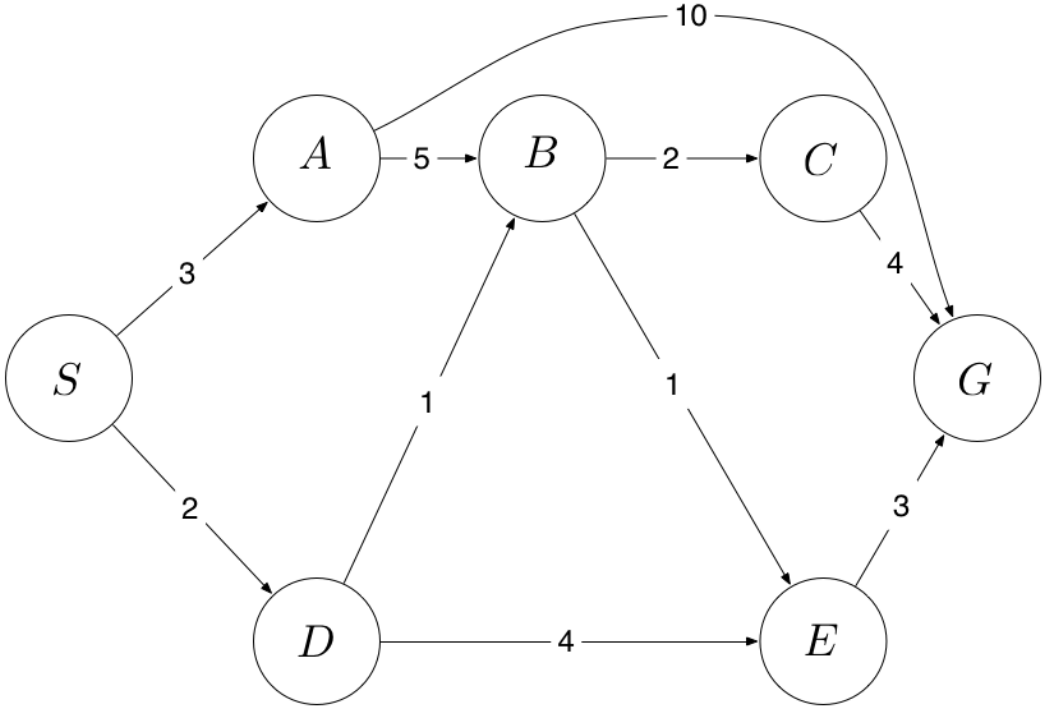




Trong khi fringe chưa rỗng (lần 8):

- Lấy kế hoạch có chi phí thấp nhất ra khỏi fringe
- Kiểm xem kế hoạch này có đi tới G?
  - Nếu có → PARTY!
  - Nếu không → mở rộng ra và đưa các kế hoạch mới vào fringe



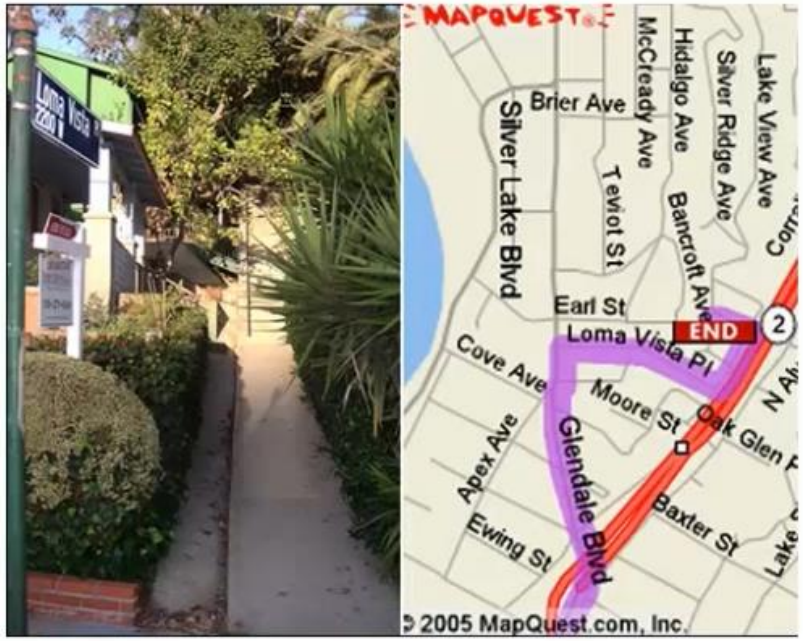


Dùng priority queue để cài đặt fringe của UCS

- Mỗi phần tử trong priority queue ứng với một kế hoạch và có độ ưu tiên là chi phí của kế hoạch (chi phí càng nhỏ thì càng được ưu tiên lấy ra trước)
- Có thể dùng priority queue để cài đặt fringe cho cả DFS và BFS, nhưng dùng stack và queue sẽ chạy nhanh hơn

Lưu ý

- Các thuật toán tìm kiếm làm việc với **bài toán tìm kiếm - mô hình về thế giới** (chỉ sau khi tìm ra kế hoạch thì mới thực thi kế hoạch này ở thế giới thực)
- Nếu mô hình sai → hy sinh (cho dù thuật toán tìm kiếm chạy đúng)



Nguồn ảnh: UC Berkely. CS188 Intro to AI

Tổng thể

- Ôn lại buổi học trước
  - Bài toán tìm kiếm
  - Các thuật toán tìm kiếm không sử dụng thông tin trạng thái đích để định hướng: DFS, BFS, UCS
- Các thuật toán tìm kiếm sử dụng thông tin trạng thái đích để định hướng

Nhìn lại về UCS

- UCS đảm bảo tìm được lời giải có chi phí nhỏ nhất
- Tuy nhiên, UCS mở rộng theo mọi hướng, không có định hướng về trạng thái đích → chạy chậm
  - Xem video demo …
- Ta mong muốn tăng tốc quá trình tìm kiếm bằng cách sử dụng thông tin trạng thái đích để định hướng