Résolution de problèmes

- Étapes intuitives par un humain
  1. modéliser la situation actuelle
  2. énumérer les solutions possibles
  3. évaluer la valeur des solutions
  4. retenir la meilleure option possible satisfaisant le but

- Mais comment parcourir efficacement la liste des solutions?

- La résolution de plusieurs problèmes peut être faite par **une recherche dans un graphe**

  - chaque nœud correspond à un état de l'environnement
  - chaque chemin à travers un graphe représente alors une suite d'actions prises par l'agent
  - pour résoudre notre problème, suffit de chercher le chemin qui satisfait le mieux notre mesure de performance

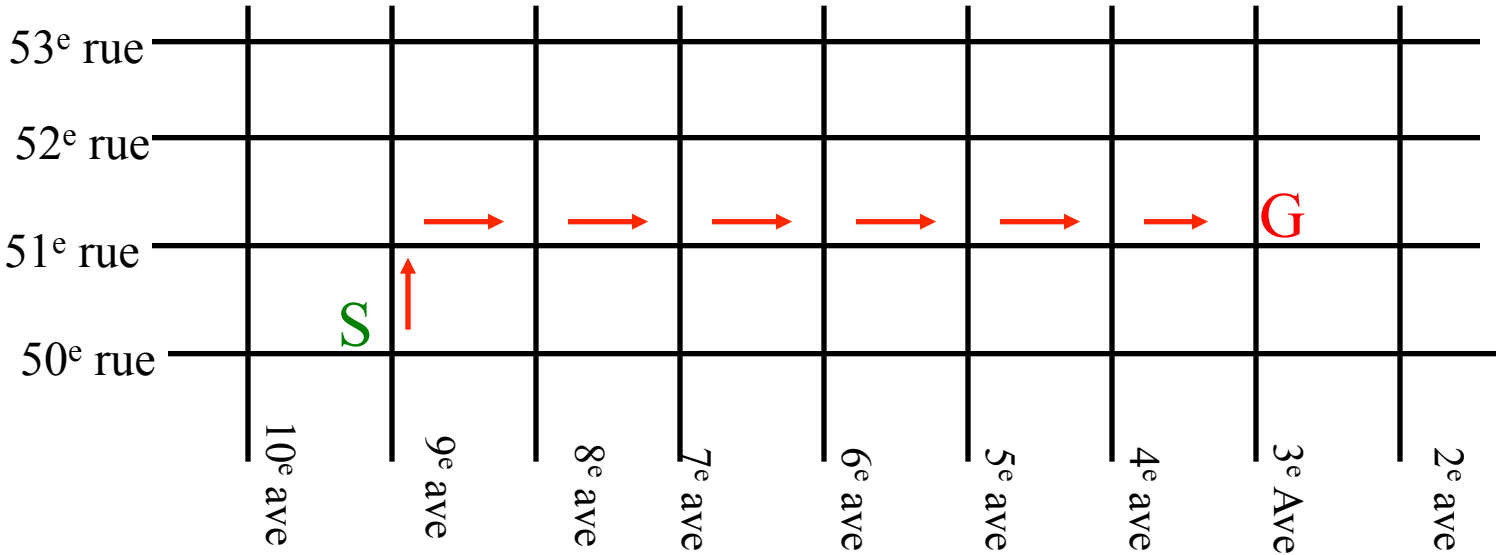
Problème de recherche dans un graphe

- Algorithme de recherche dans un graphe
  - Entrées :
    - un nœud initial
    - une fonction *goal*(*n*) qui retourne *true* si le but est atteint
    - une fonction de transition *transitions*(*n*) qui retourne les nœuds successeurs de *n*
    - une fonction *c*(*n*,*n*') strictement positive, qui retourne le coût de passer de *n* à *n*' (permet de considérer le cas avec coûts variables)
  - Sortie :
    - un chemin dans un graphe (séquence nœuds / arrêtes)
  - Le **coût d'un chemin** est la **somme des coûts des arrêtes** dans le graphe
  - Il peut y avoir plusieurs nœuds qui satisfont le but

- Enjeux :
  - trouver un chemin solution, ou
  - trouver un chemin optimal, ou
  - trouver rapidement un chemin (optimalité pas importante)

Exemple : graphe d'une ville

- Nœuds = intersections
- Arrêtes = segments de rue

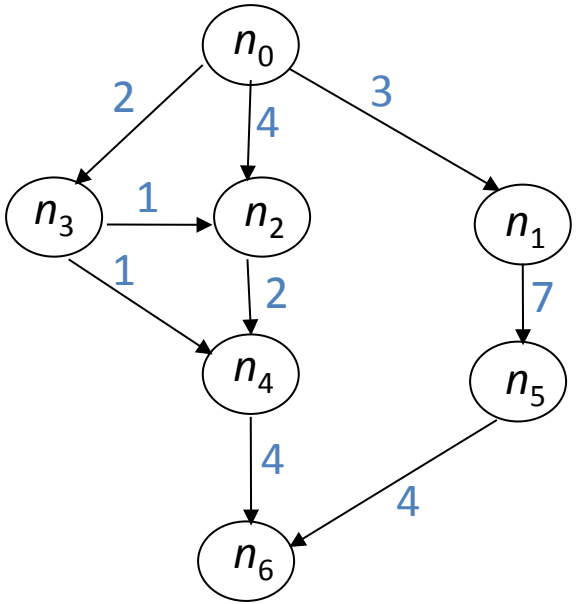


(Illustration par Henry Kautz, U. of Washington)

Exemple : trouver chemin entre deux villes

- Villes : nœuds
- Chemins entre deux villes : arrêtes

- Ville de départ : nœud (état) initial n_0
- Routes entre les villes : $transitions(n_0) = (n_3, n_2, n_1)$
- Distances entre les villes : $c(n_0,n_2) = 4$
- Ville de destination : $goal(n)$: vrai si $n=n_6$
  (où n_6 est le nœud de la ville de destination)



Rappel sur les algorithmes de recherche dans des graphes

- Recherche sans heuristique et coût uniforme
  - Recherche en profondeur (*depth-first search*)
    - » pour un nœud donné, explore le premier enfant avant d'explorer un nœud frère
  - Recherche en largeur (*breadth-first search*)
    - » pour un nœud donné, explore les nœuds frères avant leurs enfants

- Recherche sans heuristique et coût variable
  - Algorithme de Dijkstra
    - » trouve le chemin le plus court entre un nœud source et tous les autres nœuds

- Recherche avec heuristique et coût variable :
  - *best-first search*
  - *greedy best-first search*
  - **A***