Anàlisi i Disseny d'Algorismes

BACKTRACKING

Curs 2020-2021

El problema del Viatjant de Comerç

Cal implementar els dos algorismes de backtracking:

1. Backtracking Pur. Implementar un algorisme basat en backtracking que cerqui el camí òptim per fer una llista de visites del viatjant de comerç.
2. Backtracking-Greedy. Implementar un algorisme basat en backtracking i Dijkstra que cerqui el camí òptim per fer una llista de visites del viatjant de comerç.

Algorisme de backtracking pur

Espai de Cerca Cas Simple

Cerca el camí òptim d'un vèrtex V_o a un vèrtex V_d

1. Des del vèrtex V_o seguim per una aresta a un veí V_1
2. Des de V_1 anem a un veí, V_2, de V_1
3. i així successivament.

El pas endavant de l'algorisme és anar a un vèrtex veí del vèrtex actual.

En cas de no poder anar a cap veí tornem enrere i seleccionem un altre veí.

Observació: Tindrem un camí de V_o a V_d quan el vèrtex seleccionat coincideix amb el vèrtex destí V_d .

Restricció: Quan seleccionem un vèrtex, no podem passar per un vèrtex que ja hem passat.

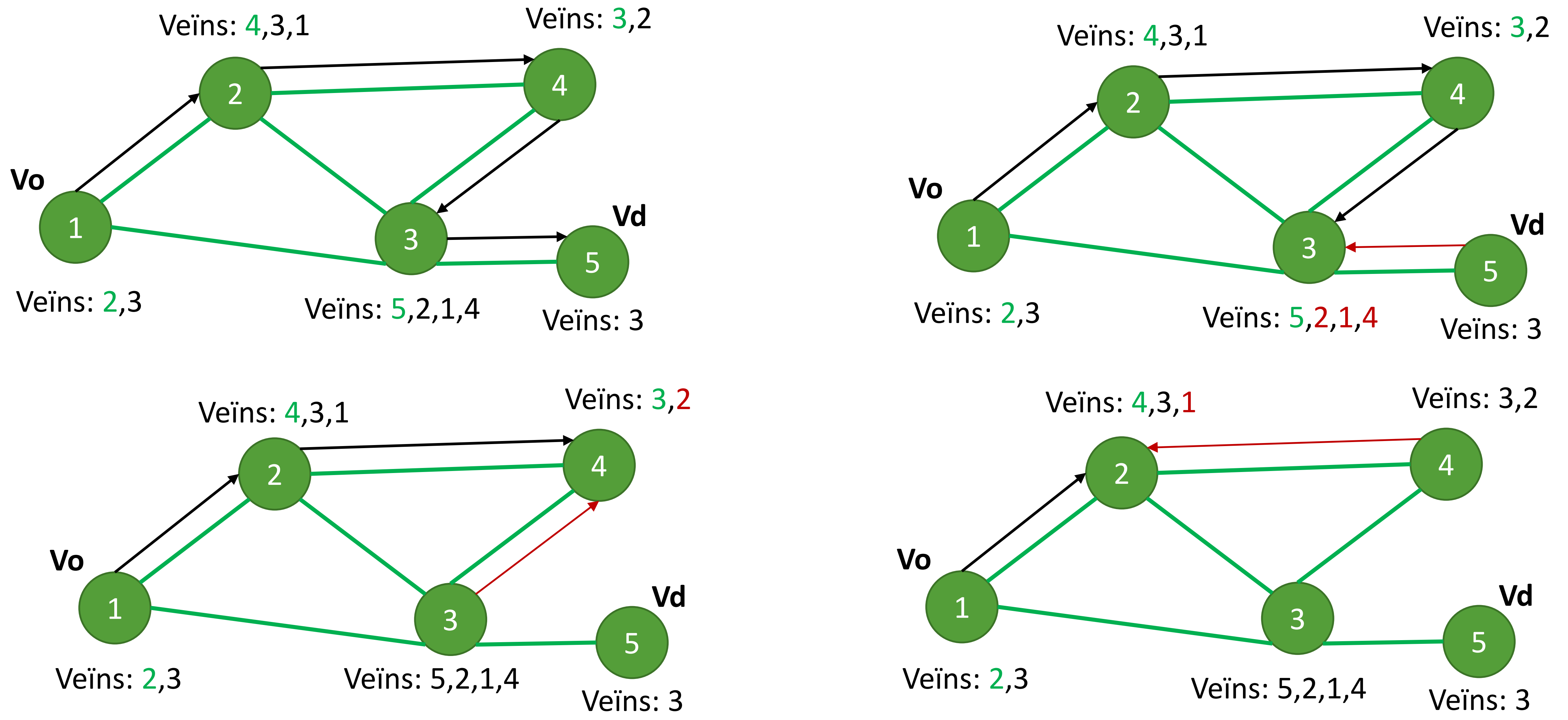
Espai de Cerca Cas Simple

Com que cerquem el camí òptim haurem de recórrer tot l'arbre d'expansió.... o no

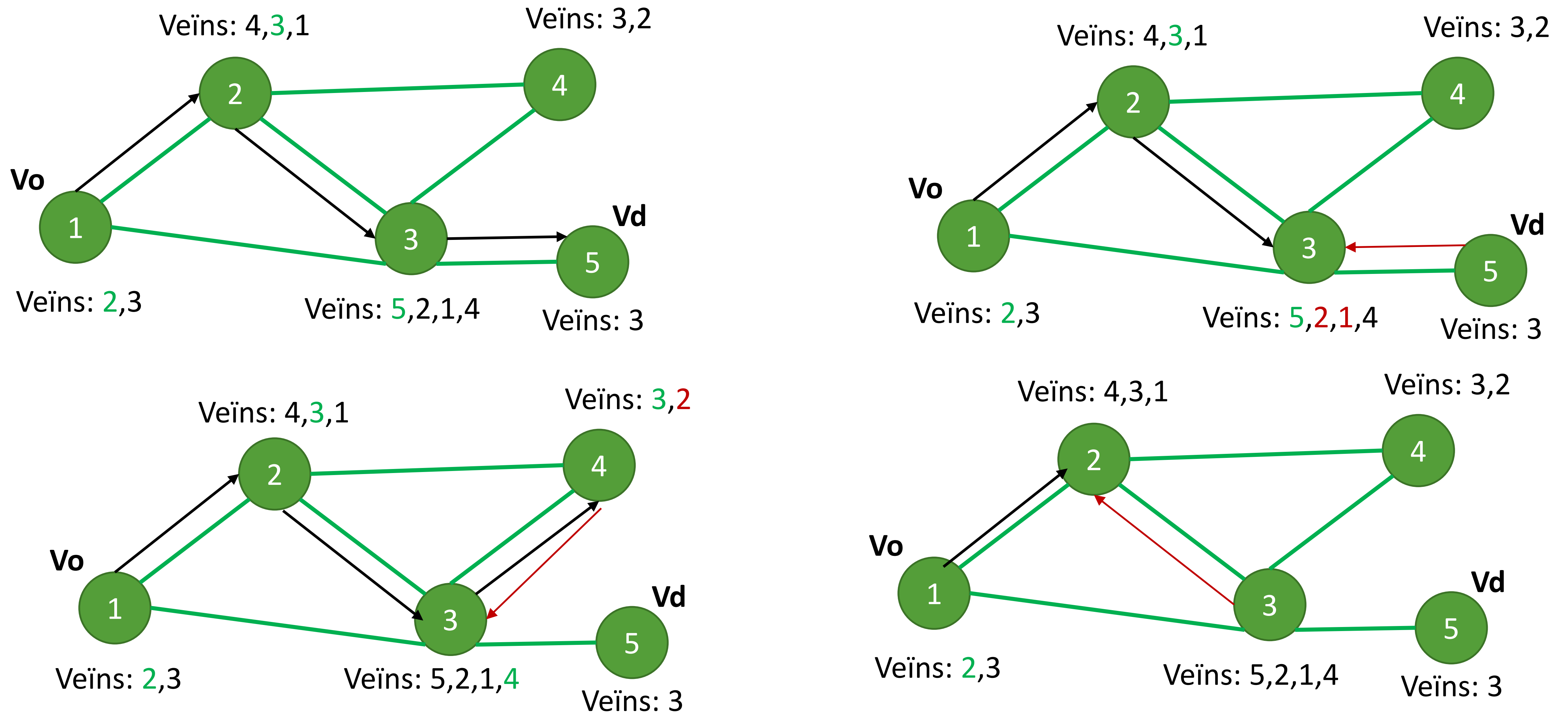
Ens podem estalviar algunes branques?

Podem mirar si el camí que estem creant és més llarg que el camí més curt que hem trobat fins el moment entre Vo i Vd

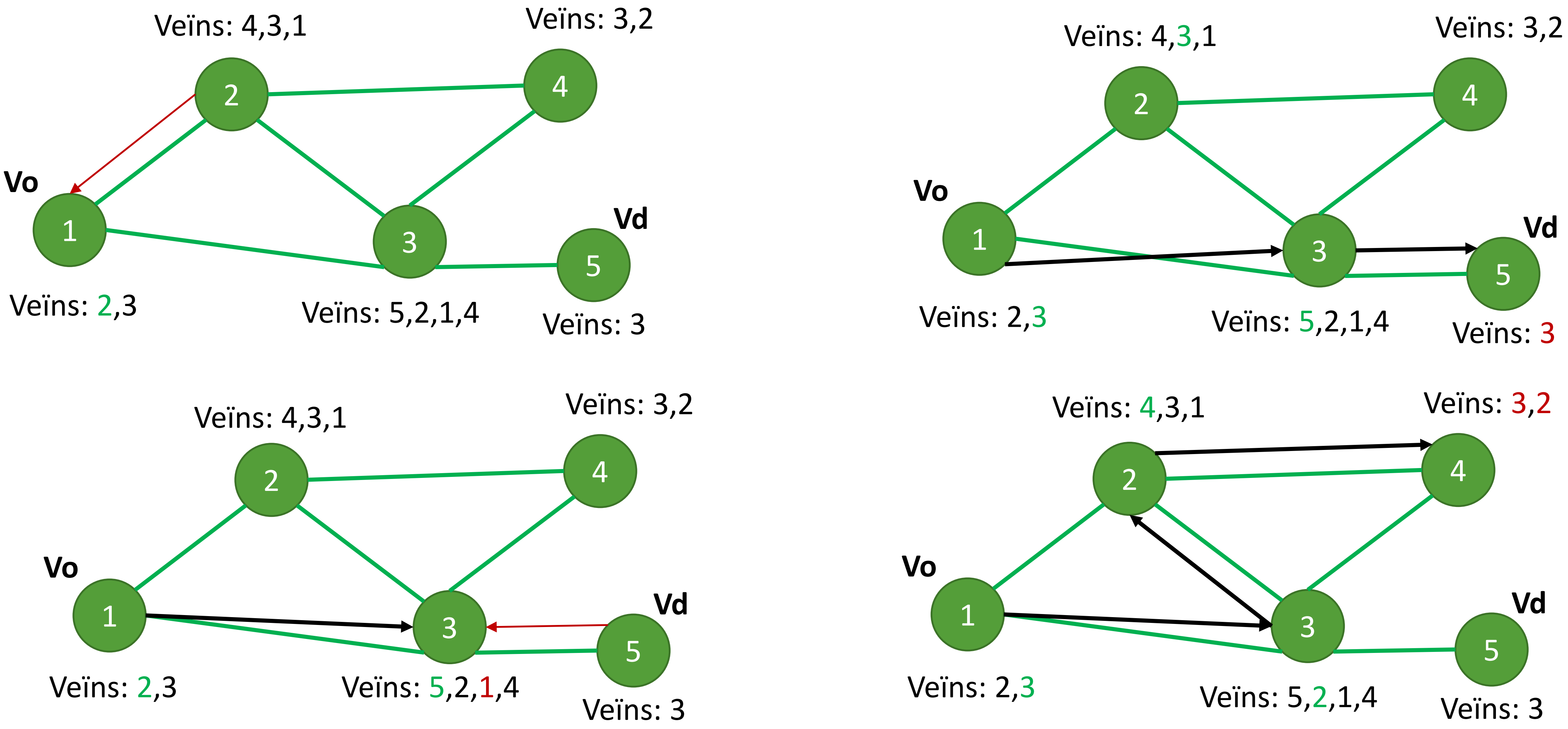
Espai de Cerca Cas Simple



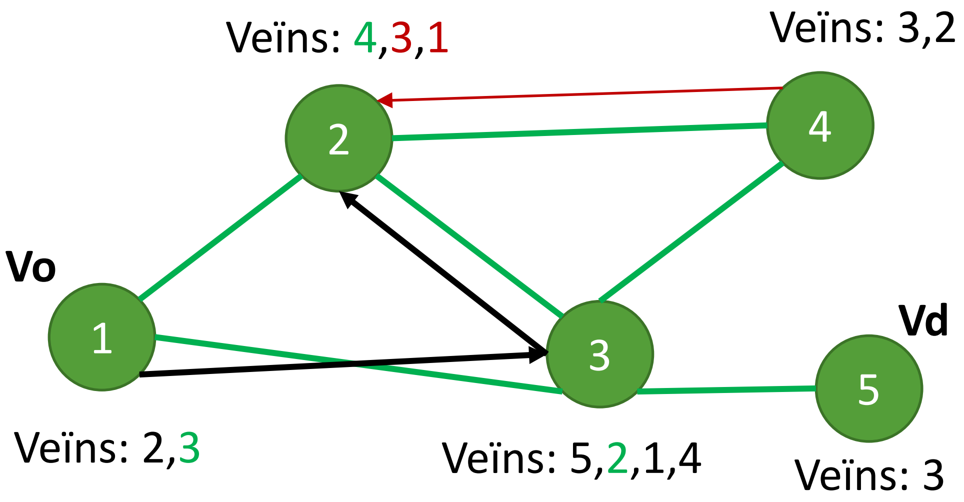
Espai de Cerca Cas Simple

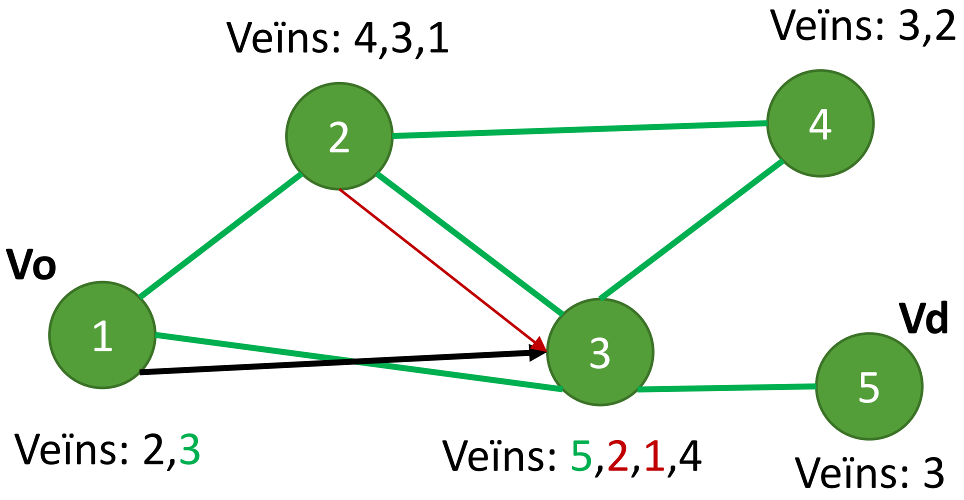


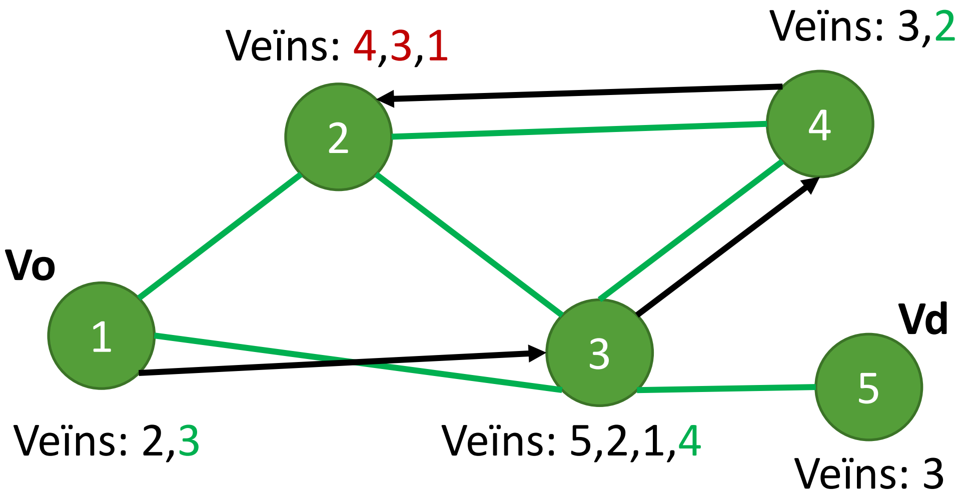
Espai de Cerca Cas Simple

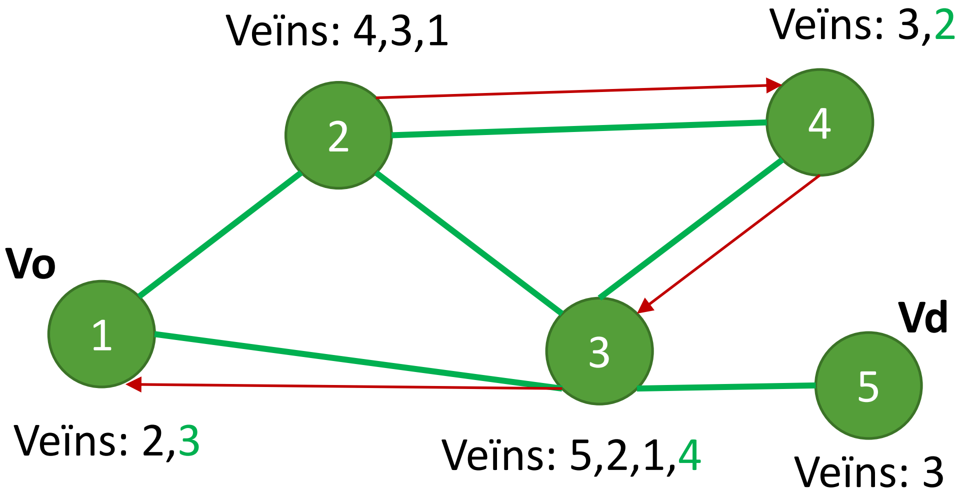


Espai de Cerca Cas Simple

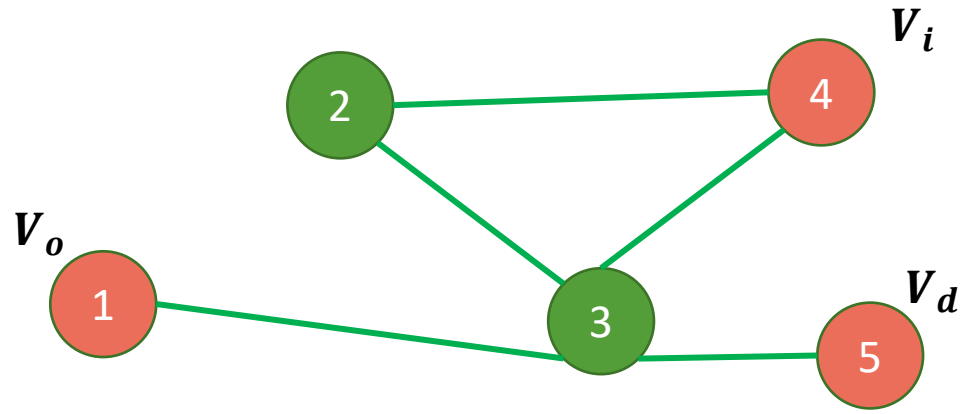








Pregunta



Podem resoldre el graf si no podem passar més d'una vegada pel 3?

Proposa un graf en el que es doni la mateixa situació.

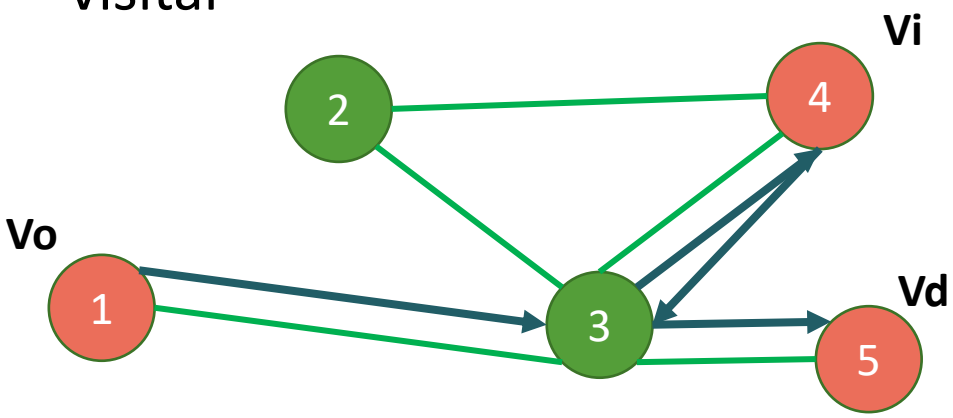
Espai de Cerca Cas Complex

- En el cas simple hem vist com es pot cercar el camí òptim entre dos vèrtexs. En el cas complex afegim que tindrem que passar per una sèrie de parades intermitges.
- Cas simple:
  - No podem passar més d'una vegada pel mateix vèrtex.
- Cas complex:
  - Sí podem passar més d'una vegada per un vèrtex si això ens porta al camí òptim.

Espai de Cerca Cas Complex

- Visites: Vo,Vi,Vd
  - El camí òptim de Vo a Vi passa per 1>3>4
  - El camí òptim de Vi a V2 passa per 4>3>5
- S'ha de repetir per força el vèrtex 3, ja que obliguem a passar pel vèrtex 4. Es passa dues vegades pel vèrtex 3, però en **camins per anar a vèrtex** diferents (Vo a Vi i Vi a Vd).
- No es poden repetir els vèrtexs que s'utilitzen per anar d'una visita B a una altra C. Si hi hagués repetició el camí no seria òptim.
- Al contrari, es pot repetir vèrtex que s'ha passat en altres camins òptims entre vèrtexs, per exemple per anar d'A a B.

- Podem veure un camí que passa per més de dos vèrtexs a visitar com la concatenació de camins òptims.
  - Camí de Vo a Vi: 1>3>4
  - Camí de Vi a Vd: 4>3>5
- Aquests camins són independents a l'hora de cercar el camí òptim que passa per tots els vèrtexs a visitar

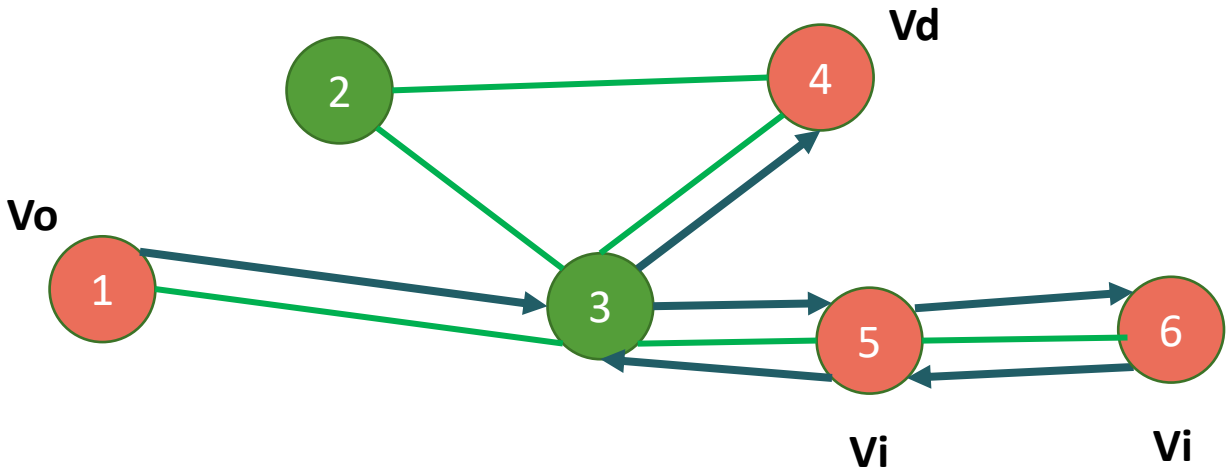


Representació del camí i de l'estat

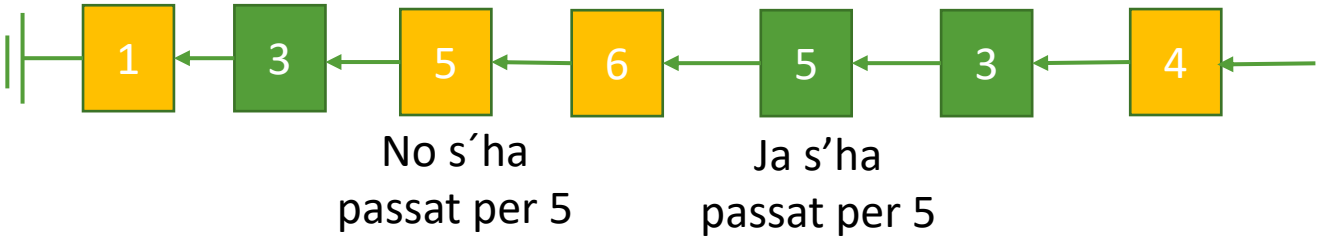
- Una possible representació per saber per quins vèrtexs hem passat és una pila
  - Posem un vèrtex a la pila quan arribem a ell en un pas endavant
  - Traiem un vèrtex de la pila quan fem un pas enrere.
- Falta controlar què passa quan arribem a un vèrtex que hem de visitar:
  - Si el vèrtex a visitar no l'havíem visitat fins ara, haurem de considerar que, al posar-lo a la pila, podem tornar a passar per tots els vèrtexs anteriors.

Representació del camí i de l'estat

- Es pot anar a un vèrtex si:
  - No es troba a la pila abans de trobar un vèrtex a visitat pel que no s'hagués passat abans.
- Camins que conformen la solució:
  - 1>3>5
  - 5>6
  - 6>5>3>4



Pila dels vèrtexs per on s'ha passat



Vèrtex a visitar pel que encara no s'ha passat.

Vèrtex que no s'ha de visitat o vèrtex a visitat pel que ja s'ha passat.

Representació del camí i de l'estat

- Pas endavant
  - Per cada veí del vèrtex actual veure si es pot passar per ell:
    - Verificar que no està a la pila abans de l'últim vèrtex a visitar pel que es passa per primera vegada (vèrtex que s'havia de visitar).
    - Si es pot passar:
      - Anar al vèrtex i posar-lo a la pila. Marcar-lo com a vèrtex que s'havia de visitar si està a la llista de vèrtexs per visitar i no s'ha passat encara per ell.
      - Crida recursiva amb el vèrtex actual (al que acabem d'anar).
    - Si no es pot passar, anar al següent veí
  - Si no hi ha més veïns, fer un pas enrere.
- Pas enrere
  - Treure un vèrtex de la pila i seleccionar com actual l'últim de la pila.

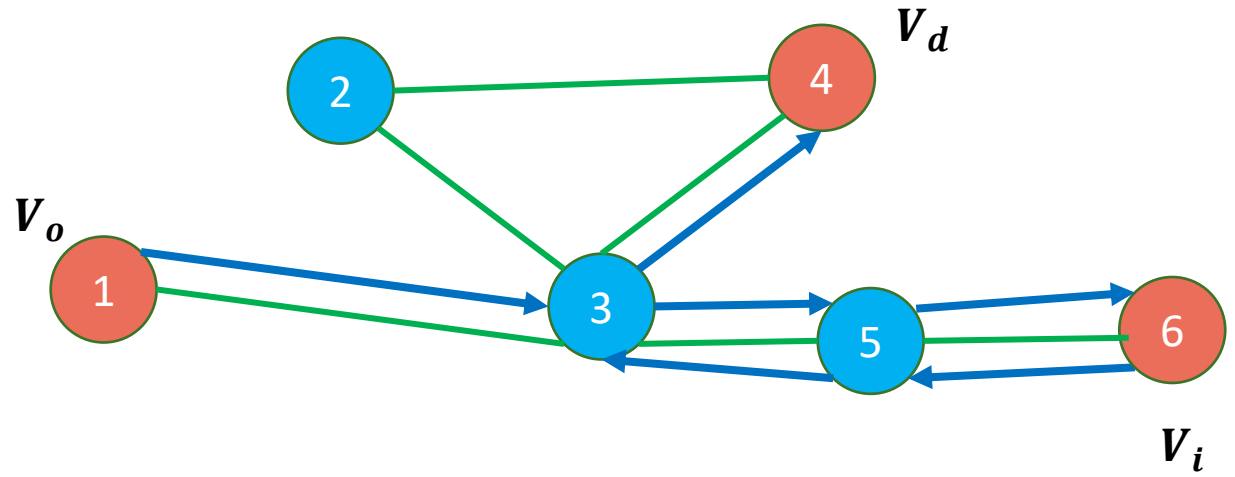
Optimitzacions de la Representació del Camí

- Evitar cercar a la llista de vèrtexs a visitar
  - Marcar els vèrtexs a visitar amb un booleà al graph. Així caldrà buscar el vèrtex a la llista de visites.
- Guardar la informació del recorregut fet en els vèrtexs del graf
  - Per guardar el recorregut fet utilitzem una pila com en el cas anterior.
  - Les dades que conté aquesta pila per cada vèrtex les incorporem en els propis vèrtexs del graf. Així evitem haver de buscar els vèrtexs a la pila per obtenir aquesta informació.
  - Per saber si un vèrtex podem passar per ell o no, considerarem en concepte de tram del camí
    - Un tram és la part del camí que va d'un vèrtex a visitar a un altre vèrtex a visitar. Així si hem de visitar n vèrtexs, el camí estarà format per n-1 trams.
    - Podem guardar a cada vèrtex el número del tram al que pertany. Compte, com podem passar més d'una vegada per cada vèrtex, aquest pot pertànyer a més d'un tram.

Backtracking sobre les arestes

Backtracking sobre arestes en graf orientat

- Aquesta forma de resoldre el problema del viatjant de comerç considera que el camí que s'està construint no pot passar més d'una vegada per la mateixa aresta en el mateix sentit.
  - Com a la representació del graf que estem utilitzant les arestes tenen sentit, això vol dir que no es pot passar més d'una vegada per la mateixa aresta.
  - En el següent esquema es pot veure que el camí que visita V_o, V_i i V_d no passa dues vegades per la mateixa aresta en el mateix sentit.



Consideracions sobre la implementació

- Per saber si s'ha passat per una aresta, podem afegir un camp a CEdge que ho indiqui.
- Solució 1: Podem cercar un camí que va del vèrtex origen al vèrtex destí de les visites sense considerar que passi per les visites intermitges.
    - Només considerarem que el camí és una solució quan estem al vèrtex destí, i després de verificar que passa per totes les visites.
- Solució 2: Podem cercar un camí que va del vèrtex origen al vèrtex destí i en la seva construcció ja comptem els passos per visites intermitges.
    - Només considerarem que el camí és una solució quan el comptador de visites intermitges és igual al total de visites i estem al vèrtex destí.
- Farem un pas de backtracking (no continuar pel mateix camí) quan:
    - Ja hem explorat totes les arestes que surten del vèrtex on acaba el camí.
    - Quan la longitud del camí actual és més gran que la del millor camí solució que hem trobat abans.
    - Quan trobem una solució.

Algorisme de backtracking-greedy

Idea bàsica

Podem:

- utilitzar l’algorisme de Dijkstra per trobar camins entre visites
- utilitzar backtracking per decidir l’ordre de les visites

Idea bàsica

- El camí òptim que passa per tots els vèrtexs a visitar està format per trams que van d'un vèrtex a visitar a un altre.
  - Els trams són camins òptims entre dos vèrtexs. L'algorisme de Dijkstra ja ens pot calcular aquests camins.
  - L'únic que haurem de decidir és en quin ordre hem de visitar els vèrtexs a visitar per aconseguir la combinació de trams que ens doni el camí òptim.
- Algorisme backtracking-greedy
  - Hem de visitar $VV=(vv_1,vv_2,...,vv_n)$
  - L'algorisme de Dijkstra ens permet calcular el camí òptim entre vv_i i vv_j qualssevol.
  - Per backtracking cerquem una tupla $(vv_1,...,vv_j,...vv_n)$ tal que la suma de les longituds dels trams de vv_j a vv_{j+1} sigui mínima.
  - El número de tuples que forma l'espai de cerca serà (n-2)!

Guies d'implementació

- Guardar en un array de dos dimensions tots els camins òptims entre vèrtexs a visitar.
  - El camí de vv_i a vv_j es guardarà en la posició (i,j) del array.
  - Cridar a Dijkstra posant com a origen cadascun dels vèrtexs a visitar. Guardar els camins generats a l'array.
- Convertir els vèrtexs a visitar en índexs d'aquest array.
- Implementar el backtracking de forma que treballi sobre un array d'índexs de vertexs (tupla).
- Tallar la cerca quan el camí que estem generant és més llarg que el camí més curt que hem trobat fins el moment.

Entrega en grup a Caronte

Entrega:

- SalesMan V2.zip amb
  - Identificació de l'alumne al fitxer SalesMan.cpp
  - Implementació dels algorismes al fitxer Backtracking.cpp.
    - `CTrack SalesmanTrackBacktracking(Cgraph& g, CVisits &visits)`
    - `CTrack SalesmanTrackBacktrackingGreedy(Cgraph& g, CVisits &visits)`
  - Projecte compilat amb mode release x64
  - Proves dels algorismes a TestSalesMan
  - Resultat correcte TrackOptimoN.txt, N de 1 a 10.
  - Abans d'entregar executar CleanProject.bat

Entrega en grup a Caronte

- Data Límit: 10/11/2020
- Entrega un sol alumne dels dos que s'identifiquen en el projecte.
- Nota:
  - Mateix mètode de càlcul de la nota que l'entrega Greedy
  - NotaEntrega=0.8*NotaTests+0.2*NotaTemps+0.5*(si els temps milloren els nostres)

  On:
    - $NotaTests$ = $\#TestsOK/\#TestsTotals$
    - $NotaTemps$ = $10 * (1 - (ta - tr)/(4tr - tr))$ o
    - ta és la suma dels temps de l'alumne de totes les proves d'una entrega o
    - tr és la suma dels temps de referència de totes les proves d'una entrega