IFT2125 - Introduction à l'algorithmique

Exploration de graphes (B&B chapitre 9)

Pierre McKenzie

DIRO, Université de Montréal

Automne 2017

Deep search of a graph (depth-first search)

Fouille en profondeur d'un graphe

B&B sections 9.3 et 9.4

revise in demo

Révisé en démo :

```
procedure dfsearch(G)
    for each v ∈ N do mark[v] ← not-visited
    for each v ∈ N do
        if mark[v] ≠ visited then dfs(v)

procedure dfs(v)
    {Node v has not previously been visited}
    mark[v] ← visited
    for each node w adjacent to v do
        if mark[w] ≠ visited then dfs(w)
```

Fouille en profondeur

Quelques utilités
Some utilities

In a non-oriented graph, the fep tree can be used to

- Dans un graphe non-orienté, l'arbre de la fep peut servir à
  - calculer les composantes connexes, / calculate the related components,
    maximum set of vertices connected two by two by a path
    ensemble maximal de sommets reliés deux à deux par un chemin
  - calculer les points d'articulation. / calculate the points of articulation.
    sommet qui, retiré, brise la connexité
    summit which, withdrawn, breaks the connection

In a directed graph, the fep tree can be used to

- Dans un graphe orienté, l'arbre de la fep peut servir à
  - détecter un cycle, detect a cycle
  - sinon à trier les sommets en ordre "topologique". / if not to sort the vertices in "topological" order.
    l'arc $s \to s'$ implique $s \leq s'$
    the arc s → s' implies s ≤ s'

- Pire cas et meilleur cas $\Theta(\max(\#\text{ d'arcs}, \#\text{ de sommets}))$.
  Worst case and best case Θ (max (# of arcs, # of vertices)).

Fouille en largeur d'un graphe

B&B section 9.5

```
procedure bfs(v)
    Q ← empty-queue
    mark[v] ← visited
    enqueue v into Q
    while Q is not empty do
        u ← first(Q)
        dequeue u from Q
        for each node w adjacent to u do
            if mark[w] ≠ visited then mark[w] ← visited
                enqueue w into Q
```

In both cases we need a main program to start the search.

```
procedure search(G)
    for each v ∈ N do mark[v] ← not-visited
    for each v ∈ N do
        if mark[v] ≠ visited then {dfs2 or bfs} (v)
```

Fouille en largeur

Utilité
Utility

- Trouvera le sommet recherché, dans un graphe infini de degré borné, si un tel sommet existe.

  Will find the desired vertex, in an infinite graph of bounded degree, if such a vertex exists.

Fouille d'un graphe par retour arrière (backtracking)

B&B Section 9.6

Context

Contexte :

a graph that is often implicit because it is too big or even infinite

- graphe souvent implicite, car trop grand ou même infini
- souvent sans cycle, même un arbre often without a cycle, even a tree
- sommet = solution partielle top = partial solution
- recherché : sommet qui est solution complète.

  searched: top that is complete solution.

L'idée : étendre constamment une solution partielle et rebrousser chemin dès la détection de l'absence de solution complète le long d'une branche.

The idea: to constantly extend a partial solution and to turn back when the absence of complete solution is detected along a branch.

Retour arrière

Forme générale
General form

```
procedure backtrack(v[1..k])
    {v is a k-promising vector}
    if v is a solution then write v
    {else} for each (k + 1)-promising vector w
                such that w[1..k] = v[1..k]
                do backtrack(w[1..k + 1])
```

Note : w n'est pas "weight" mais simplement un vecteur qui prolonge v.

Note: w is not "weight" but simply a vector that extends v.

Retour arrière : exemple 1

Backpack with multiplicites

Sac à dos avec multiplicités

capacity and types of objects 1, 2, . . . , n of weight

DONNÉE: capacité $W \in \mathbb{R}^{\geq 0}$ et types d'objets $1, 2, \ldots, n$ de poids

given and of values

$w_1, \ldots, w_n \in \mathbb{R}^{\geq 0}$ et de valeurs $v_1, \ldots, v_n \in \mathbb{R}^{\geq 0}$

CALCULER: comme d'habitude mais avec les $x_i \in \mathbb{N}$

calculate as usual but with the x_i in N

Le retour arrière ressemble à la fouille en profondeur :

Backtracking is like depth-first search:

```
function backpack(i, r)
    {Calculates the value of the best load that can
     be constructed using items of types i to n
     and whose total weight does not exceed r}
    b ← 0
    {Try each allowed kind of item in turn}
    for k ← i to n do
        if w[k] ≤ r then
            b ← max(b, v[k] + backpack(k, r − w[k]))
    return b
```

Initial call

Appel initial : $backpack(1, W)$.

Objets 1,2,3,4 de valeurs 3,5,6,10 et poids 2,3,4,5, capacité 8.

Arbre typique d'un algo de retour arrière :

Objects 1,2,3,4 of values 3,5,6,10 and weight 2,3,4,5, capacity 8.

Typical tree of a backtrack algo:

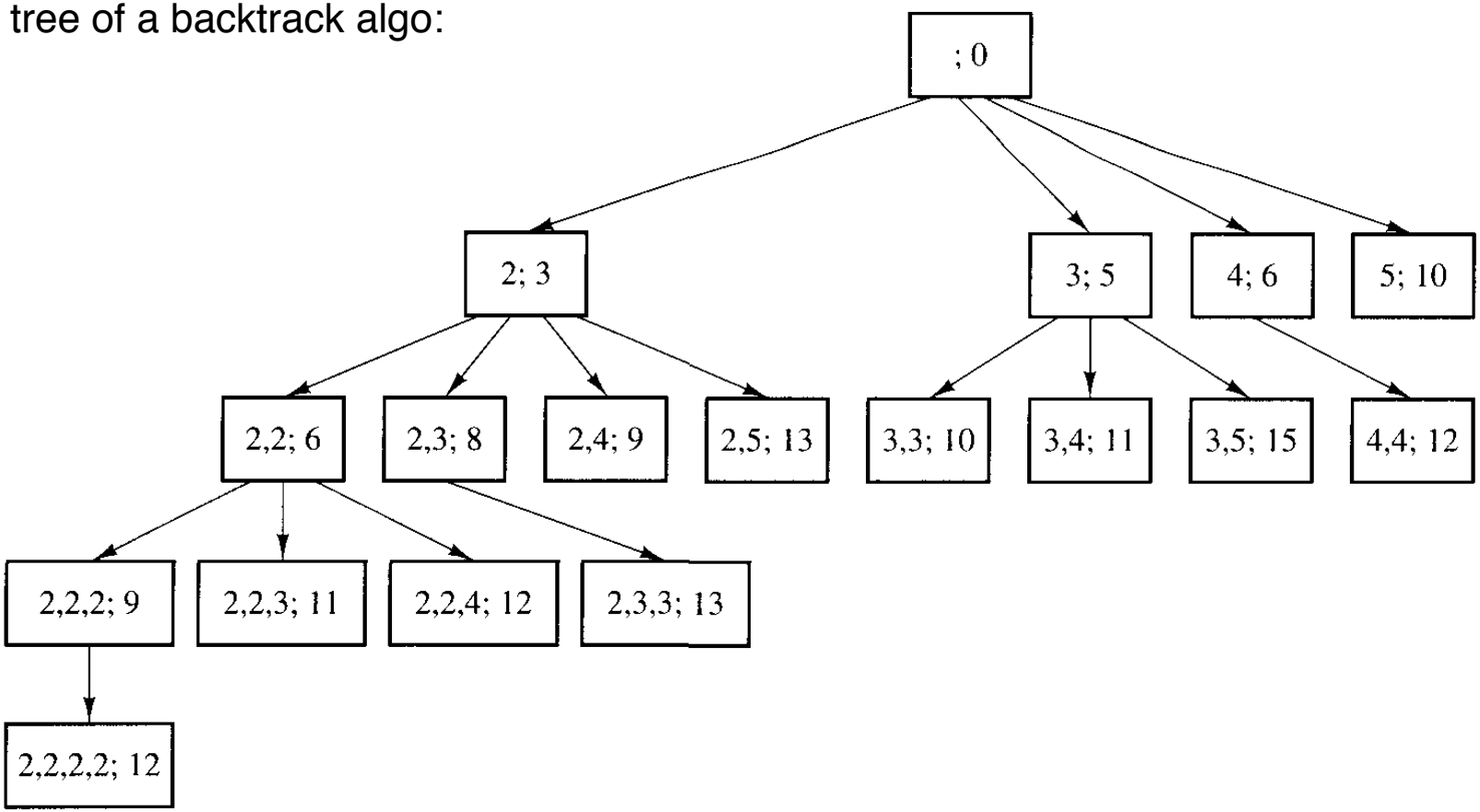

Figure 9.12. The implicit tree for a knapsack problem

2, 2, 4; 12 represents objects 1,1,3 (weight 2 and 2 and 4) totaling value 12

2, 2, 4; 12 représente objets 1,1,3 (poids 2 et 2 et 4) totalisant valeur 12

Retour arrière : exemple 2

Can we place 8 queens on the game without 2 queens in the pinch?

Peut-on placer 8 reines sur le jeu sans que 2 reines ne soient en prise ?

En classe.

In class

Search by "separation and evaluation" (branch and bound)

Fouille par “séparation et évaluation” (branch and bound)

B&B Section 9.6

Context: each vertex is solution but one seeks the optimal one.

Contexte : chaque sommet est solution mais on cherche l’optimale.

L’idée : estimer pour chaque sommet visité une valeur de “favorabilité” et explorer ensuite les branches paraissant les plus favorables.

The idea: to estimate for each summit visited a value of "favorability" and then to explore the branches appearing the most favorable.

- raffinement du retour-arrière
- programmation inélégante car ni en profondeur, ni en largeur
- difficile et souvent impossible à analyser de manière théorique.

- Rewind refinement
- inelegant programming because neither in depth nor in width
- difficult and often impossible to analyze theoretically.

Principe du minimax

B&B Section 9.8

Context:
- implicit graph of a game with two players (ex: chess)
- vertex = game configuration (eg positioning of the pieces)
- arc s1 → s2 = possible move from s1 to s2
- each s receives a value v (s) of "favorability towards the player A"
- Wanted: A good shot of Starting from s
- heuristic only because v (s) imperfect.

Contexte :

- graphe implicite d'un jeu à deux joueurs (ex : échecs)
- sommet = configuration du jeu (ex : positionnement des pièces)
- arc $s_1 \to s_2$ = coup possible de s_1 vers s_2
- chaque s reçoit une valeur $v(s)$ de "favorabilité envers le joueur A"
- recherché : un bon coup de A partant de s
- heuristique seulement car $v(s)$ imparfaite.

Principe du minimax

(suite)
(cont.)

The idea: A good shot of A from s is to play to s1 if

L'idée : un bon coup de A à partir de s est de jouer vers s_1 si

- $s \rightarrow s_1$ et and
  $v(s_1) = \max_{s \rightarrow s'} \{v(s')\}$,
- or better still ou mieux encore $s \rightarrow s_1 \rightarrow s_2$ et and
  $v(s_2) = \max_{s \rightarrow s'} \{\min_{s' \rightarrow s''} \{v(s'')\}\}$,
- or better still ou mieux encore $s \rightarrow s_1 \rightarrow s_2 \rightarrow s_3$ et and
  $v(s_3) = \max_{s \rightarrow s'} \{\min_{s' \rightarrow s''} \{\max_{s'' \rightarrow s'''} \{v(s''')\}\}\}$
- et ainsi de suite selon puissance de calcul disponible !
  and so on according to available computing power!