Maîtrise Informatique 2003 / 2004

UE11
Devoir d'algorithmique

Julien Van Den Bossche / Benoît Moulin

Généralités pour le devoir :
Les algorithmes sont programmés en Haskell. Pour construire les cycles que l'on doit générés on utilise une liste qui peut, être assimilée à une pile, c'est-à-dire que le premier élément du cycle est le dernier de cette pile.
Dans les algorithmes décris ci-dessous on ne tient pas compte de ces « inversions ». Leur prise en compte se fera dans le code du programme Haskell.
La notation $A_i(X,Y)$ signifie que A_i est une arête composée des sommets X et Y.
La notation *premier(liste),* signifie que l'on prend le premier élément de la liste.

Les codes sources des programmes se trouvent sur www.julienvdb.com/algo

1. Procédure backtrack

La procédure backtrack est un algorithme qui prend en donnée un entier n positif, des ensembles D_i, appelés domaines, qui sont au nombre de n, ainsi qu'une application C_i qui est définie par :

Pour tout i ($1 \leq i \leq n$), $C_i : D_1 \times \ldots \times D_i \rightarrow \{vrai, faux\}$.

Cette procédure génère toutes les suites $(\sigma(1),\ldots, \sigma(n))$ tel que, pour i ($1 \leq i \leq n$), $\sigma(1) \in D_i$ et $C_i(\sigma(1), \ldots, \sigma(i)) = vrai$, σ étant une application :

$$\sigma : \{1, \ldots, n\} \rightarrow \{ \sigma(1),\ldots, \sigma(n)\}.$$

Algorithme

Début :
$i \leftarrow 1$; $\Delta_1 \leftarrow D_1$

Tant que : $i \geq 1$ faire
Début{tant que}
Si $\Delta_i \neq \emptyset$
Alors [soit $a \in \Delta_i$;
$\sigma(i) \leftarrow a$;
$\Delta_i \leftarrow \Delta_i - \{a\}$;
Si $C_i(\sigma(1),\ldots, \sigma(i)) = vrai$
Alors [si i=n alors ecrire (σ) sinon [$i \leftarrow i + 1$; $\Delta_i \leftarrow D_i$]]
]
sinon $i \leftarrow i - 1$
fin{tant que}
FIN.

2. Cycles hamiltoniens

Généralités pour les cycles hamiltoniens :

Toutes les listes de sommets sont triées par ordre croissant.
Pour tenir compte du filtrage des solutions dans les conditions Ci nous avons procédé comme suit :

> Si le deuxième élément du cycle est plus petit que le dernier (sachant qu'il est connexe avec l'élément de départ) cela signifie que le cycle n'a pas été visité.

Exemple : 1, 4, 5, 3, 4, 2, 3, 1, 2, 5 (cycle 1) est correcte car 5 > 4
En revanche si on a : 1, 5, 2, 1, 3, 2, 4, 3, 5, 4 (cycle 2) on ne le prendra pas car c'est une permutation du cycle 1. On a bien 4 < 5.
On peut procéder ainsi pour détecter nos permutations car on a considéré nos listes de sommets triées par ordre croissant.

2.1. Utilisation de la procédure Backtrack

On utilise la définition proposée à la question 1 pour générer les cycles hamiltoniens d'un graphe donné G(S,A).
Pour qu'un cycle, dans un graphe, soit hamiltonien on doit vérifier que le cycle passe par toutes les arêtes du graphe et que chaque sommet voisin dans le cycle forme une arête de G.

Identification des variables par rapport au problème :

-Soit n le nombre de sommets du graphe.

-Soit $D = \{D_1 \times \ldots\ldots \times D_n\}$ le domaine tel que $1 \leq i \leq n$, D_i = liste des sommets adjacents au sommet numéro i.

-Conditions (Ci) sur les sommets Si :

> Pour tout $1 \leq i, j \leq n$ $S_i \neq S_j$ //tous les sommets sont différents.
> (S_i, S_{i+1}) forme une arête de G.
> $S_2 < S_n$. //Pour éviter les mêmes cycles, on regarde les permutations.

Algorithme :

Initialisation

Soit *départ* le sommet de départ du parcours du graphe, *départ <- 1*.
Soit *i*, l'indice qui nous indiquera que l'on empile le $i^{ième}$ sommet dans le cycle, *i<-départ, 1 ≤ i ≤ n.*
Soit *P*, une pile représentant un cycle en construction, *P = [i]*.
On considère que toutes les listes d'adjacences sont ordonnées par sommet croissant.
Soit Δ_i = la liste des sommets adjacents au $i^{ème}$ sommet du cycle en construction.
Cette liste contient les sommets non visités.
Au départ $\Delta_i = D_i$.

Boucle

Tant que la pile n'est pas vide (la pile est vide quand i = 0)

Si i == n et que $D_{\text{tête de pile}}$ contient départ alors
Si tête de pile < deuxième élément empilé. //P appartient aux cycles créés
i <- i-1 //retour en arrière
Δ_i <- Δ_i / {tête de pile}
On dépile P
Sinon
On ajoute la pile renversée à la liste des cycles.
i <- i-1
Δ_i <- Δ_i / {tetedepile}
On dépile P
Sinon
Si Δ_i n'est pas vide
S_{choisi} <- premier sommet de Δ_i
Si S_{choisi} n'appartient pas à la pile
On empile S_{choisi} dans P
Δ_i <- Δ_i / {S_{choisi}}
i <- i+1
Δ_i <- Sommet connexe à S_{choisi}
Sinon Δ_i <- Δ_i / {S_{choisi}}

Sinon
Soit s = tête de pile.

On dépile P
i <- i-1
Δ_i <- Δ_i / {s}

Fin tant que

Résultats obtenus :

On obtient 30 cycles hamiltoniens différents :

[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20],
[1,2,3,4,5,12,11,10,6,7,8,9,19,18,17,13,14,15,16,20],
[1,2,3,4,5,12,11,18,17,13,14,15,16,20,19,9,10,6,7,8],
[1,2,3,4,5,12,13,14,15,16,17,18,11,10,6,7,8,9,19,20],
[1,2,3,4,14,15,16,17,13,12,5,6,7,8,9,10,11,18,19,20],
[1,2,3,4,14,15,16,20,19,9,10,11,18,17,13,12,5,6,7,8],
[1,2,3,7,6,5,4,14,15,16,20,19,18,17,13,12,11,10,9,8],
[1,2,3,7,6,10,11,18,17,13,12,5,4,14,15,16,20,19,9,8],
[1,2,3,7,8,9,10,6,5,4,14,15,16,17,13,12,11,18,19,20],
[1,2,3,7,8,9,19,18,17,13,12,11,10,6,5,4,14,15,16,20],
[1,2,15,14,4,3,7,6,5,12,13,17,16,20,19,18,11,10,9,8],
[1,2,15,14,4,3,7,8,9,19,18,11,10,6,5,12,13,17,16,20],
[1,2,15,14,13,12,5,4,3,7,6,10,11,18,17,16,20,19,9,8],
[1,2,15,14,13,12,11,10,6,5,4,3,7,8,9,19,18,17,16,20],
[1,2,15,14,13,12,11,18,17,16,20,19,9,10,6,5,4,3,7,8],
[1,2,15,14,13,17,16,20,19,18,11,12,5,4,3,7,6,10,9,8],
[1,2,15,16,17,13,14,4,3,7,8,9,10,6,5,12,11,18,19,20],
[1,2,15,16,17,18,11,10,6,5,12,13,14,4,3,7,8,9,19,20],
[1,2,15,16,20,19,9,10,6,5,12,11,18,17,13,14,4,3,7,8],
[1,2,15,16,20,19,18,17,13,14,4,3,7,6,5,12,11,10,9,8],
[1,8,7,3,2,15,14,4,5,6,10,9,19,18,11,12,13,17,16,20],
[1,8,7,3,2,15,16,17,18,11,12,13,14,4,5,6,10,9,19,20],
[1,8,7,6,5,4,3,2,15,14,13,12,11,10,9,19,18,17,16,20],
[1,8,7,6,5,12,11,10,9,19,18,17,13,14,4,3,2,15,16,20],
[1,8,7,6,5,12,13,14,4,3,2,15,16,17,18,11,10,9,19,20],
[1,8,7,6,10,9,19,18,11,12,5,4,3,2,15,14,13,17,16,20],
[1,8,9,10,6,7,3,2,15,16,17,13,14,4,5,12,11,18,19,20],
[1,8,9,10,11,12,13,14,4,5,6,7,3,2,15,16,17,18,19,20],
[1,8,9,19,18,11,10,6,7,3,2,15,14,4,5,12,13,17,16,20],
[1,8,9,19,18,17,13,14,4,5,12,11,10,6,7,3,2,15,16,20].

3. Cycles eulériens

Généralités sur les cycles eulériens :

Toutes les arêtes sont triées par sommet croissant. On prendra (2,3) au lieu de (3,2).
Pour tenir compte du filtrage des solutions dans les conditions C_i nous avons procédé de la même manière que dans les cycles hamiltoniens en regardant tous les sous-cycles d'un cycle.

Exemple : 1, 3, 2, 1, 4, 2, 5, 3, 4, 5.

En rouge nous retrouvons un sous cycle et ce dernier a déjà été parcouru dans l'autre sens : 1, 2, 3, 1, 4, 2, 5, 3, 4, 5.
En effet, on essaye de mettre toujours le sommet le plus petit en premier dans notre algorithme.

Dans cette partie nous travaillons avec des arêtes mais il faut générer des sommets.
Le cycle en construction contenant alors des sommets, il est plus facile de tester nos conditions sur le filtrage, dans le code Haskell, avec des sommets que de tester avec des arêtes.

Par contre, pour savoir si une arête est déjà présente dans le cycle nous serons obligé de passer par une fonction, dans le code Haskell, qui transforme la liste de sommets en liste d'arêtes.

3.1. Montrer que $X_1 = Y_q$

G est un graphe eulérien, ce qui signifie qu'il a un cycle eulérien. Celui-ci passe une et une seule fois par chaque arête du graphe, et, comme tout cycle, le dernier élément correspond au point de départ du cycle. La dernière arête doit aboutir au sommet d'origine.
La première arête du graphe est $\{X_1,Y_1\}$, d'après la condition (ii), donc le sommet de départ est X_1. La dernière arête est $\{X_q, Y_q\}$ car le nombre d'arêtes de G est q, d'après (i). L'arête $\{X_q, Y_q\}$ doit donc aboutir au sommet d'origine. On en déduit alors que le dernier sommet du cycle, Y_q, correspond à celui de départ, X_1.

Donc on a $\mathbf{X_1 = Y_q}$

3.2. Utilisation de la procédure backtrack

Identification des variables par rapport au problème :

-Soit *n* le nombre d'arêtes différentes du graphe.
-Soit $D = \{D_1 \times \ldots\ldots \times D_n\}$, $D_1 = D_i = D_n =$ liste des différentes arêtes du graphe.
-Conditions (C_i) sur les arêtes A_i

Pour tout $1 \leq i, j < n$, $A_i(X_i, Y_i) \neq A_j(X_j, Y_j)$
Si $X_i == Y_j$ alors $X_j > Y_i$ //filtrage des sous cycles avec leurs permutations.
Si i == n-1 alors $Y_1 < Y_{n-1}$

Algorithme

Initialisation

Soit *départ* le sommet de départ du parcours du graphe, *départ <- 1.*
Soit *i*, l'indice qui indiquera que l'on empile le $i^{ième}$ sommet dans le cycle, *i<- départ, $1 \leq i \leq n$.*
Soit *P*, une pile représentant un cycle en construction, *P = [i]*.
On considère que toutes les arêtes sont ordonnées par sommet croissant.
Soit Δ_i = la liste des arêtes partant du $i^{ème}$ sommet du cycle en construction.
Cette liste contient les arêtes non visitées.
Au départ Δ_i = ensemble des arêtes partant du sommet de départ (sommet 1).

Boucle

Tant que la pile n'est pas vide (la pile est vide quand i = 0)

Si i == n alors
Si tetedepile < deuxième élément empilé. //P appartient aux cycles créés
Soit s1 = tête de pile
On dépile P.
Soit s2 = tête de pile
Soit A = (s1, s2).
Δ_i <- Δ_i / {A}

Sinon

On ajoute la pile renversée à la liste des cycles.
Soit s1 = tête de pile
On dépile P.
Soit s2 = tête de pile
Soit A = (s1, s2).
i <- i-1
Δ_i <- Δ_i / {A}

Sinon

Si Δ_i n'est pas vide
$A_{choisie}$ <- premier Δ_i
Si $A_{choisie}$ n'appartient pas à la pile
SommetAEmpiler = Xi si tête de pile = Yj sinon Yj
On empile *sommetAEmpiler* dans P
Δ_i <- Δ_i / {$A_{choisie}$}
i <- i+1
Δ_i <- Sommet connexe à *sommetAEmpiler*

Sinon Δ_i <- Δ_i / { $A_{choisie}$ }

Sinon
Soit s1 = tête de pile
On dépile P.
Soit s2 = tête de pile
Soit A = (s1, s2).
i <- i-1
Δ_i <- Δ_i / {A}

Fin tant que

Résultats obtenus

Avec notre filtre on ne trouve plus que 5 cycles eulériens différents.
En prenant en compte que les permutations sur l'ensemble du cycle et en ne s'occupant pas des sous cycles nous trouvons 264 cycles.
Nous n'arrivons pas à bien déterminer la différence entre deux cycles.

[1, 2, 3, 1, 4, 2, 5, 3, 4, 5],
[1, 2, 3, 1, 5, 2, 4, 3, 5, 4],
[1, 2, 3, 4, 1, 5, 2, 4, 5, 3],
[1, 2, 4, 1, 5, 2, 3, 4, 5, 3],
[1, 2, 4, 3, 1, 5, 2, 3, 5, 4].

4. Arbres

4.1. Gp-1 est un arbre

G est un graphe connexe à p sommets, il a donc comme ensemble d'arêtes $\{\{X_1,Y_1\}, \ldots, \{X_p,Y_p\}\}$. G_{p-1} est le graphe qui a le même ensemble de sommet que G et $\{\{X_1, Y_1\}, \ldots, \{X_{p-1}, Y_{p-1}\}\}$ comme ensemble d'arêtes. Par hypothèse (énoncé), aucun chemin de G_{p-1} ne relie X_p et Y_p. Donc, s'il y a un début de cycle, il est impossible d'aboutir sur le sommet d'origine. Ce qui signifie que le graphe G_{p-1} est sans cycle.
G_{p-1} est un graphe connexe sans cycle, et ceci étant la définition d'un arbre, on en déduit que G_{p-1} est un arbre.

4.2. Utilisation de la procédure backtrack

Initialisation

Soit ***n*** le nombre d'arêtes différentes du graphe.
Soit ***départ*** le sommet de départ du parcours du graphe, ***départ <- 1***.
Soit ***i***, l'indice qui indiquera que l'on empile la $i^{ième}$ arête dans l'arbre, $1 \leq i \leq n-1$.
Soit ***P***, une pile représentant un arbre en construction, ***P = []***.
Soit $D= \{D_1 \times \ldots\ldots\ldots \times D_n\}$, $D_1 = D_i = D_n =$ liste des différentes arêtes du graphe.
On considère que toutes les arêtes sont ordonnées par sommet croissant.
Soit ***res*** = l'ensemble des sommets visités, au départ ***res*** contient le sommet ***départ***.
Soit ***L*** la liste de toutes les arêtes possibles du graphe : pour tout $\{Xi,Yi\} \in D$, $1 \leq i \leq n$, ***L*** contient $\{X_i,Y_i\}+\{Y_i,X_i\}$.
Soit Δ_i = la liste des arêtes qui peuvent succéder à la $i^{ème}$ arête de l'arbre en construction.
Plus précisément $\Delta i = L/\{X_j,Y_i\}$ si X_j et $Y_j \in$ ***res***, ce qui permet d'éviter les cycles ou si X_j et Y_j n'appartiennent pas à ***res*** ce qui permet d'enlever les arêtes non possibles (arêtes ayant aucune racine).
On utilisera creeDelta(***res***) qui décrit ces actions dans l'algo ci-dessous.

Conditions (Ci) sur les arêtes Ai

Pour tout $1 \leq i, j < n$, A_i (X_i,Y_i) = arête que l'on doit tester.
$[(X_1,Y_1),\ldots, (X_{i-1},Y_{i-1})]$ l'arbre en construction.

Si l'arbre est vide alors *Vrai*
Si $X_i == Y_{i-1}$ alors *Vrai.*
Si $X_i < X_{i-1}$ *alors*
    Si $X_i == Y_{i-1}$ alors *Vrai.*
    Sinon *Faux.*
Si $X_i == X_{i-1}$ alors
    Si $Y_i < Y_{i-1}$ *Faux.* //On a déjà généré cet arbre car on parcourt nos arêtes par ordre croissant.
    Sinon on teste notre arête avec l'arbre $[(X_1,Y_1),\ldots, (X_{i-2},Y_{i-2})]$

Sinon on teste notre arête avec l'arbre $[(X_1,Y_1),\ldots, (X_{i-2},Y_{i-2})]$

Boucle

Tant que i > 1

    Si i == n – 1 alors
        On renvoi l'arbre obtenu
        Soit A = tête de pile
        On dépile P.
        i <- i-1
        Δ_i <- Δ_i / {A}

    Sinon

        Si Δ_i n'est pas vide
        $A_{choisie}$ (X,Y) <- premier Δ_i
            Si Condition($A_{choisie}$, pile) = vrai
                Empile($A_{choisie}$)
                Res <- res+nouveauSommet($A_{choisie}$) //Un des sommets de $A_{choisie}$ qui n'appartient pas à *res*
                i <- i+1
                Δ_i <- creeDelta(*res*).
            Sinon Δ_i <- Δ_i / { $A_{choisie}$ }

        Sinon

Soit A = tête de pile
depile(P).
i <- i-1
Δ_i <- Δ_i / {A}

Fin tant que

Résultats obtenus

En partant du sommet 1, nous trouvons 125 arbres de recouvrements :

[(1,2) (1,3) (1,4) (1,5)]	[(1,2) (2,4) (2,5) (4,3)]	[(1,3) (3,2) (3,5) (5,4)]
[(1,2) (1,3) (1,4) (2,5)]	[(1,2) (2,4) (2,5) (5,3)]	[(1,4) (1,5) (4,2) (2,3)]
[(1,2) (1,3) (1,4) (3,5)]	[(1,2) (2,4) (4,5) (5,3)]	[(1,4) (1,5) (4,2) (4,3)]
[(1,2) (1,3) (1,4) (4,5)]	[(1,2) (2,4) (4,3) (3,5)]	[(1,4) (1,5) (4,2) (5,3)]
[(1,2) (1,3) (1,5) (2,4)]	[(1,2) (2,4) (4,3) (4,5)]	[(1,4) (1,5) (5,2) (2,3)]
[(1,2) (1,3) (1,5) (3,4)]	[(1,2) (2,5) (5,3) (3,4)]	[(1,4) (1,5) (5,2) (5,3)]
[(1,2) (1,3) (1,5) (5,4)]	[(1,2) (2,5) (5,3) (5,4)]	[(1,4) (1,5) (4,3) (3,2)]
[(1,2) (1,3) (2,4) (2,5)]	[(1,2) (2,5) (5,4) (4,3)]	[(1,4) (1,5) (4,3) (5,2)]
[(1,2) (1,3) (2,4) (3,5)]	[(1,3) (1,4) (1,5) (3,2)]	[(1,4) (1,5) (5,3) (3,2)]
[(1,2) (1,3) (2,4) (4,5)]	[(1,3) (1,4) (1,5) (4,2)]	[(1,4) (4,5) (5,2) (2,3)]
[(1,2) (1,3) (2,5) (3,4)]	[(1,3) (1,4) (1,5) (5,2)]	[(1,4) (4,5) (5,2) (5,3)]
[(1,2) (1,3) (2,5) (5,4)]	[(1,3) (1,4) (3,5) (4,2)]	[(1,4) (4,5) (5,3) (3,2)]
[(1,2) (1,3) (3,4) (3,5)]	[(1,3) (1,4) (3,5) (5,2)]	[(1,4) (4,2) (2,3) (2,5)]
[(1,2) (1,3) (3,4) (4,5)]	[(1,3) (1,4) (4,5) (5,2)]	[(1,4) (4,2) (2,3) (3,5)]
[(1,2) (1,3) (3,5) (5,4)]	[(1,3) (1,4) (3,2) (2,5)]	[(1,4) (4,2) (2,3) (4,5)]
[(1,2) (1,4) (1,5) (2,3)]	[(1,3) (1,4) (3,2) (3,5)]	[(1,4) (4,2) (2,5) (4,3)]
[(1,2) (1,4) (1,5) (4,3)]	[(1,3) (1,4) (3,2) (4,5)]	[(1,4) (4,2) (2,5) (5,3)]
[(1,2) (1,4) (1,5) (5,3)]	[(1,3) (1,4) (4,2) (2,5)]	[(1,4) (4,2) (4,5) (5,3)]
[(1,2) (1,4) (2,3) (2,5)]	[(1,3) (1,4) (4,2) (4,5)]	[(1,4) (4,2) (4,3) (3,5)]
[(1,2) (1,4) (2,3) (3,5)]	[(1,3) (1,5) (3,4) (4,2)]	[(1,4) (4,2) (4,3) (4,5)]
[(1,2) (1,4) (2,3) (4,5)]	[(1,3) (1,5) (3,4) (5,2)]	[(1,4) (4,3) (3,5) (5,2)]
[(1,2) (1,4) (2,5) (4,3)]	[(1,3) (1,5) (3,2) (2,4)]	[(1,4) (4,3) (4,5) (5,2)]
[(1,2) (1,4) (2,5) (5,3)]	[(1,3) (1,5) (3,2) (3,4)]	[(1,4) (4,3) (3,2) (2,5)]
[(1,2) (1,4) (4,5) (5,3)]	[(1,3) (1,5) (3,2) (5,4)]	[(1,4) (4,3) (3,2) (3,5)]
[(1,2) (1,4) (4,3) (3,5)]	[(1,3) (1,5) (5,2) (2,4)]	[(1,4) (4,3) (3,2) (4,5)]
[(1,2) (1,4) (4,3) (4,5)]	[(1,3) (1,5) (5,2) (5,4)]	[(1,5) (5,2) (2,3) (2,4)]
[(1,2) (1,5) (2,3) (2,4)]	[(1,3) (1,5) (5,4) (4,2)]	[(1,5) (5,2) (2,3) (3,4)]
[(1,2) (1,5) (2,3) (3,4)]	[(1,3) (3,4) (3,5) (4,2)]	[(1,5) (5,2) (2,3) (5,4)]
[(1,2) (1,5) (2,3) (5,4)]	[(1,3) (3,4) (3,5) (5,2)]	[(1,5) (5,2) (2,4) (4,3)]
[(1,2) (1,5) (2,4) (4,3)]	[(1,3) (3,4) (4,5) (5,2)]	[(1,5) (5,2) (2,4) (5,3)]
[(1,2) (1,5) (2,4) (5,3)]	[(1,3) (3,4) (4,2) (2,5)]	[(1,5) (5,2) (5,3) (3,4)]

[(1,2) (1,5) (5,3) (3,4)]
[(1,2) (1,5) (5,3) (5,4)]
[(1,2) (1,5) (5,4) (4,3)]
[(1,2) (2,3) (2,4) (2,5)]
[(1,2) (2,3) (2,4) (3,5)]
[(1,2) (2,3) (2,4) (4,5)]
[(1,2) (2,3) (2,5) (3,4)]
[(1,2) (2,3) (2,5) (5,4)]
[(1,2) (2,3) (3,4) (3,5)]
[(1,2) (2,3) (3,4) (4,5)]
[(1,2) (2,3) (3,5) (5,4)]

[(1,3) (3,4) (4,2) (4,5)]
[(1,3) (3,5) (5,2) (2,4)]
[(1,3) (3,5) (5,2) (5,4)]
[(1,3) (3,5) (5,4) (4,2)]
[(1,3) (3,2) (2,4) (2,5)]
[(1,3) (3,2) (2,4) (3,5)]
[(1,3) (3,2) (2,4) (4,5)]
[(1,3) (3,2) (2,5) (3,4)]
[(1,3) (3,2) (2,5) (5,4)]
[(1,3) (3,2) (3,4) (3,5)]
[(1,3) (3,2) (3,4) (4,5)]

[(1,5) (5,2) (5,3) (5,4)]
[(1,5) (5,2) (5,4) (4,3)]
[(1,5) (5,3) (3,4) (4,2)]
[(1,5) (5,3) (3,2) (2,4)]
[(1,5) (5,3) (3,2) (3,4)]
[(1,5) (5,3) (3,2) (5,4)]
[(1,5) (5,3) (5,4) (4,2)]
[(1,5) (5,4) (4,2) (2,3)]
[(1,5) (5,4) (4,2) (4,3)]
[(1,5) (5,4) (4,3) (3,2)]