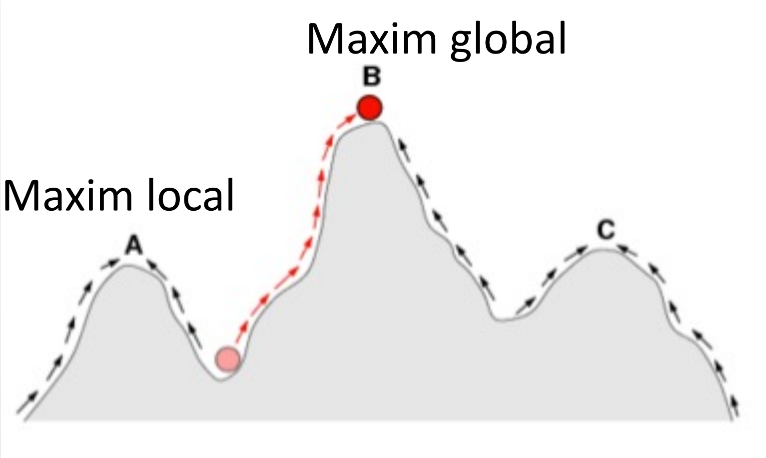
Curs 10 SDA: Tehnica GREEDY

Definitie si consideratii generale
Exemple: Selectia activitatilor. Problema numararii restului. Problema rucsacului. Coduri Huffman. TSP.
Greedy vs backtracking – analiza si exemple

Metoda greedy – consideratii generale

- Se aplica in **probleme de optimizare combinatoriala** – au ca solutii submultimi sau elemente ale unor produse carteziene pentru care se alege optimul (minimul sau maximul) functiei obiectiv.
- Determina intotdeauna *o singura solutie* a problemei
- Solutia este construita treptat:
  - Initial solutia este vida
  - Se alege pe rand elementul cel mai promitator corespunzator situatiei la momentul curent – se aleg elementele care asigura un optim local - fapt care nu garanteaza o solutie globala optima !
  - Optimalitatea solutiei aleasa de greedy se demonstreaza.
    - Se foloseste demonstatia prin inductie.
      Daca se gaseste un contra-exemplu atunci solutia nu este optima !



http://databuckets.blogspot.ro/2016/01/decision-making-in-terms-of-search.html

Metoda greedy – consideratii generale

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

- Este o metoda **simpla**
- Programele care o implementeaza pot avea performante bune chiar in cazul unor date de dimensiuni mari.
- Simplitatea metodei este data de faptul ca la fiecare pas se considera doar o componenta a solutiei, folosind criterii locale de selectie

Greedy – forma generala

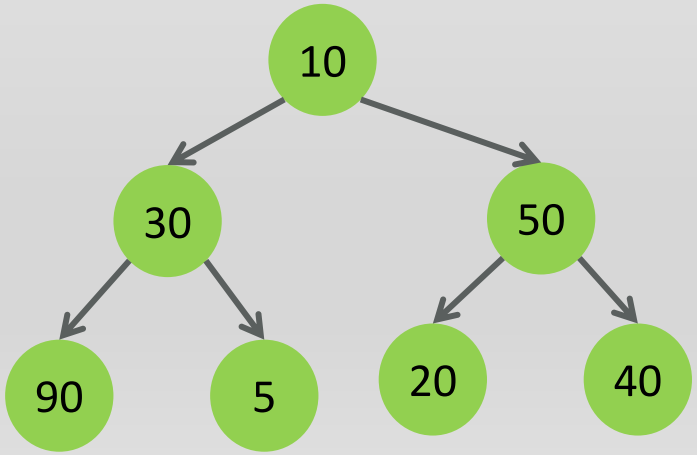
Fie S solutia problemei.

Fie C multimea din care se aleg elementele componente ale solutiei.

Greedy(C)
S←∅
While S is not a solution and C ≠∅ **do**
    choose x the most promising element in C
    remove x from C
    if it is possible add x to S

If S is a solution **then**
    process solution (print solution)
    else message(no solution was found)

Exemplu – determinati suma maxima pe ramura



Greedy – forma generala

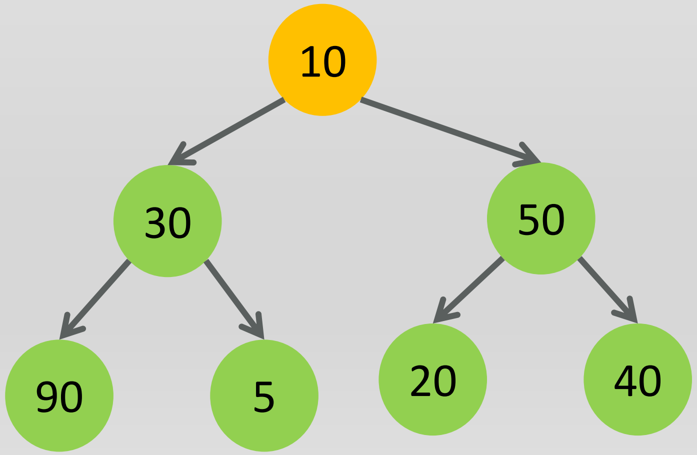
Fie S solutia problemei.

Fie C multimea din care se aleg elementele componente ale solutiei.

Greedy(C)
S←Ø
While S is not a solution and C ≠Ø **do**
    choose x the most promising element in C
    remove x from C
    if it is possible add x to S

If S is a solution **then**
    process solution (print solution)
    else message(no solution was found)

Exemplu – determinati suma maxima pe ramura



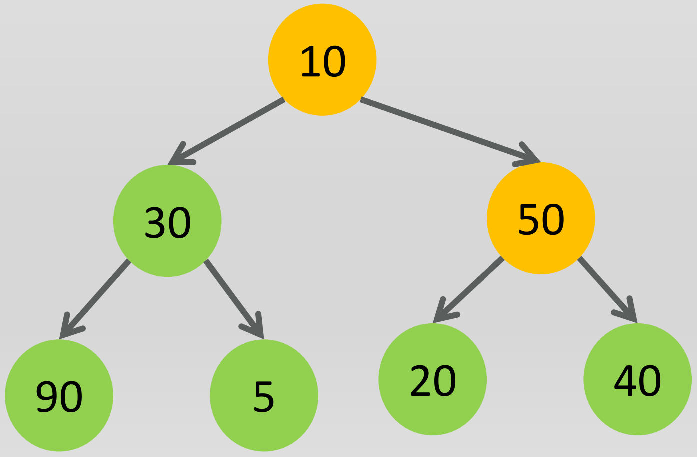
Greedy – forma generala

Fie S solutia problemei.
Fie C multimea din care se aleg elementele componente ale solutiei.

```
Greedy(C)
S←Ø
While S is not a solution and C ≠Ø do
        choose x the most promising element in C
        remove x from C
        if it is possible add x to S

If S is a solution then
        process solution (print solution)
        else message(no solution was found)
```

Exemplu – determinati suma maxima pe ramura



Greedy – forma generala

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

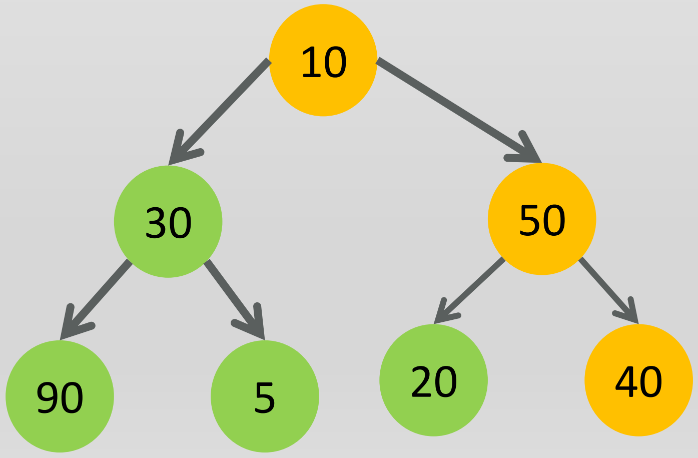
Fie S solutia problemei.

Fie C multimea din care se aleg elementele componente ale solutiei.

Greedy(C)
S←∅
While S is not a solution and C ≠∅ **do**
    choose x the most promising element in C
    remove x from C
    if it is possible add x to S

If S is a solution **then**
    process solution (print solution)
    else message(no solution was found)

Exemplu – determinati suma maxima pe ramura



Suma este 100 ! Este suma maxima?

Greedy – forma generala

Fie S solutia problemei.

Fie C multimea din care se aleg elementele componente ale solutiei.

Greedy(C)

S←Ø

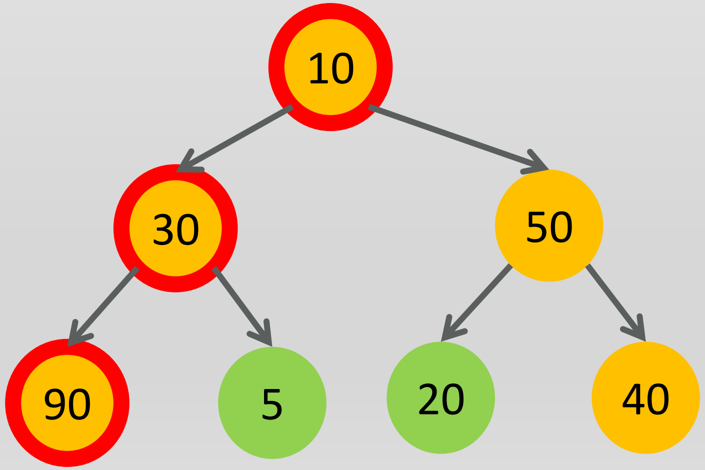
While S is not a solution and C ≠Ø **do**

    choose x the most promising element in C
    remove x from C
    if it is possible add x to S

If S is a solution **then**

    process solution (print solution)
    else message(no solution was found)

Exemplu – determinati suma maxima pe ramura

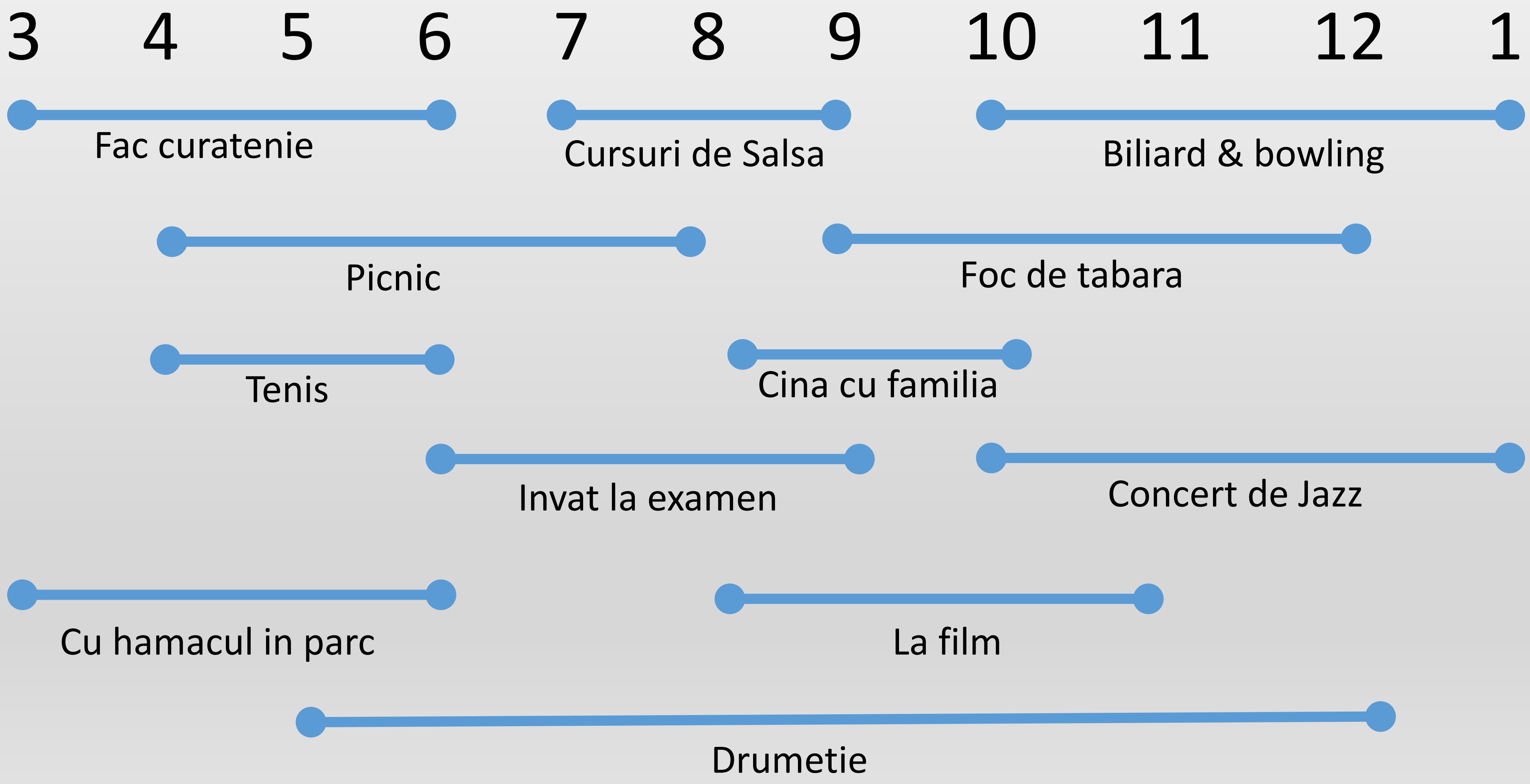


Solutia mai buna determina suma = 130

Tehnica greedy

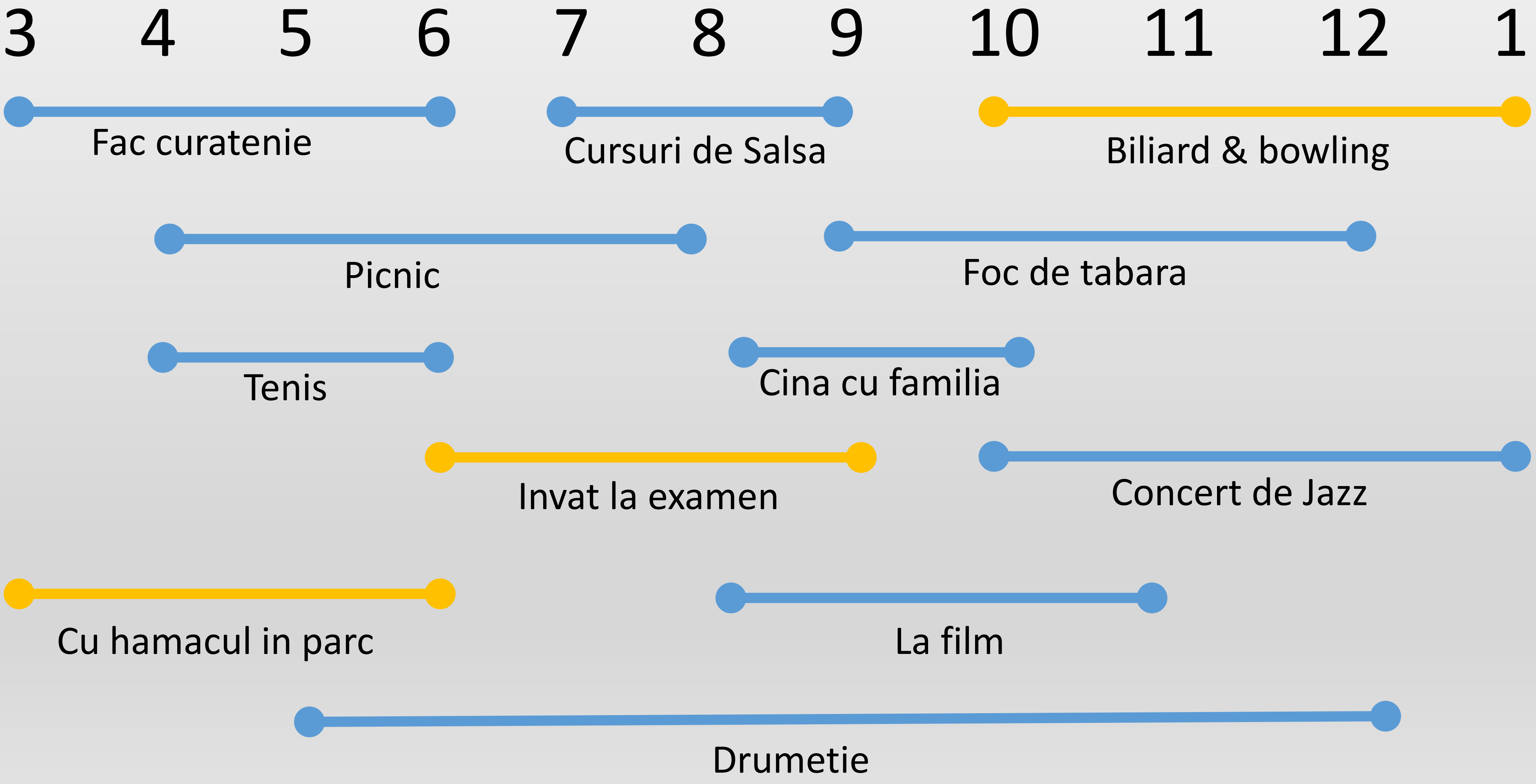
- Un algoritm **greedy** selecteaza mereu alternativa cea mai buna la acel moment, in speranta ca aceea va duce la solutia optima globala.
- Nu garanteaza gasirea solutiei optime, dar exista situatii in care acest lucru este posibil:
  - problema selectiei activitatilor
  - arborele minim de acoperire – Minimum Spanning Tree (Prim, Kruskall)
  - gasirea cailor de cost minim in graf (Dijkstra)
  - etc.

Selectia activitatilor – sa planificam o zi de weekend !

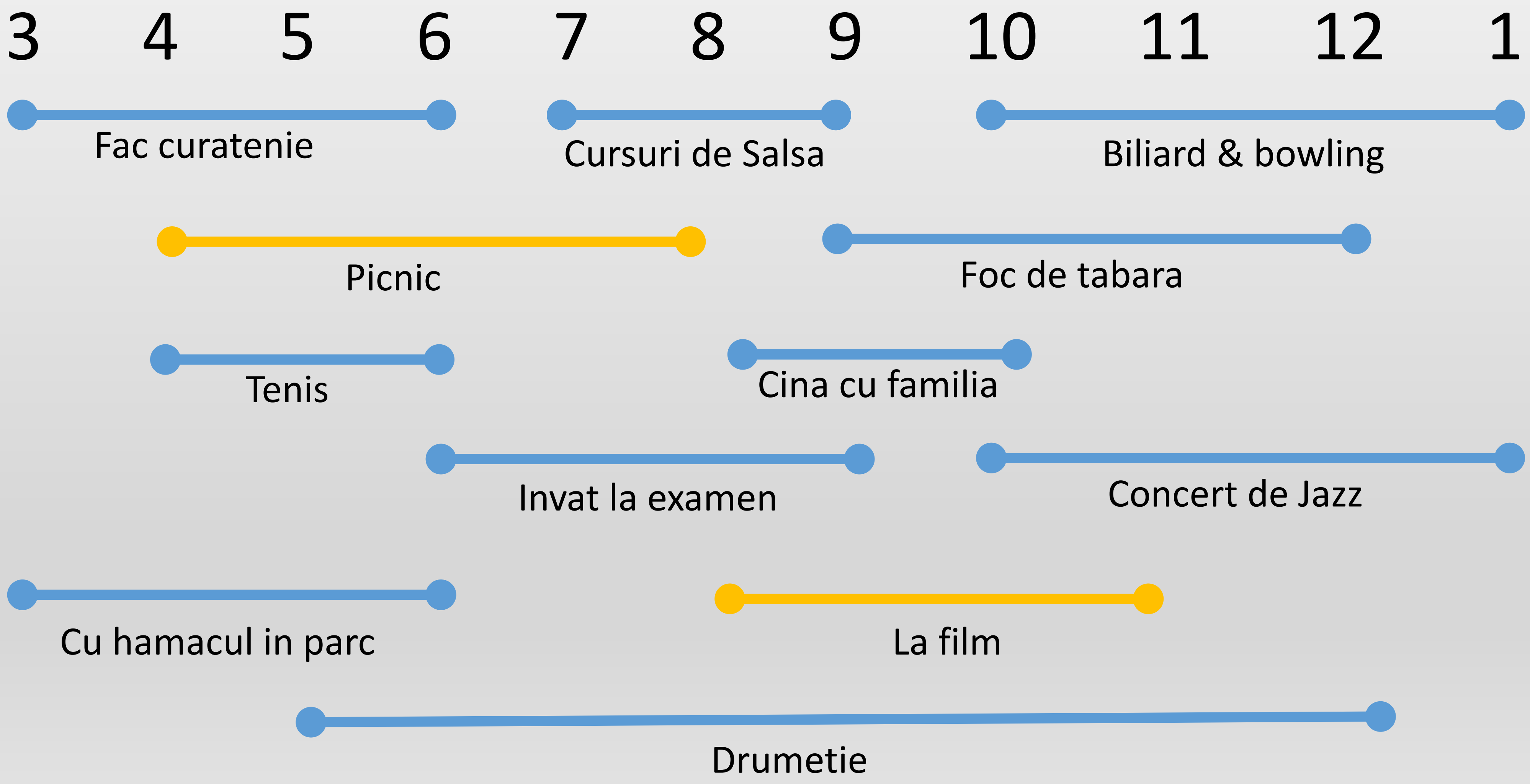


Selectia activitatilor –
sa planificam o zi de weekend !

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA



Selectia activitatilor –
sa planificam o zi de weekend !



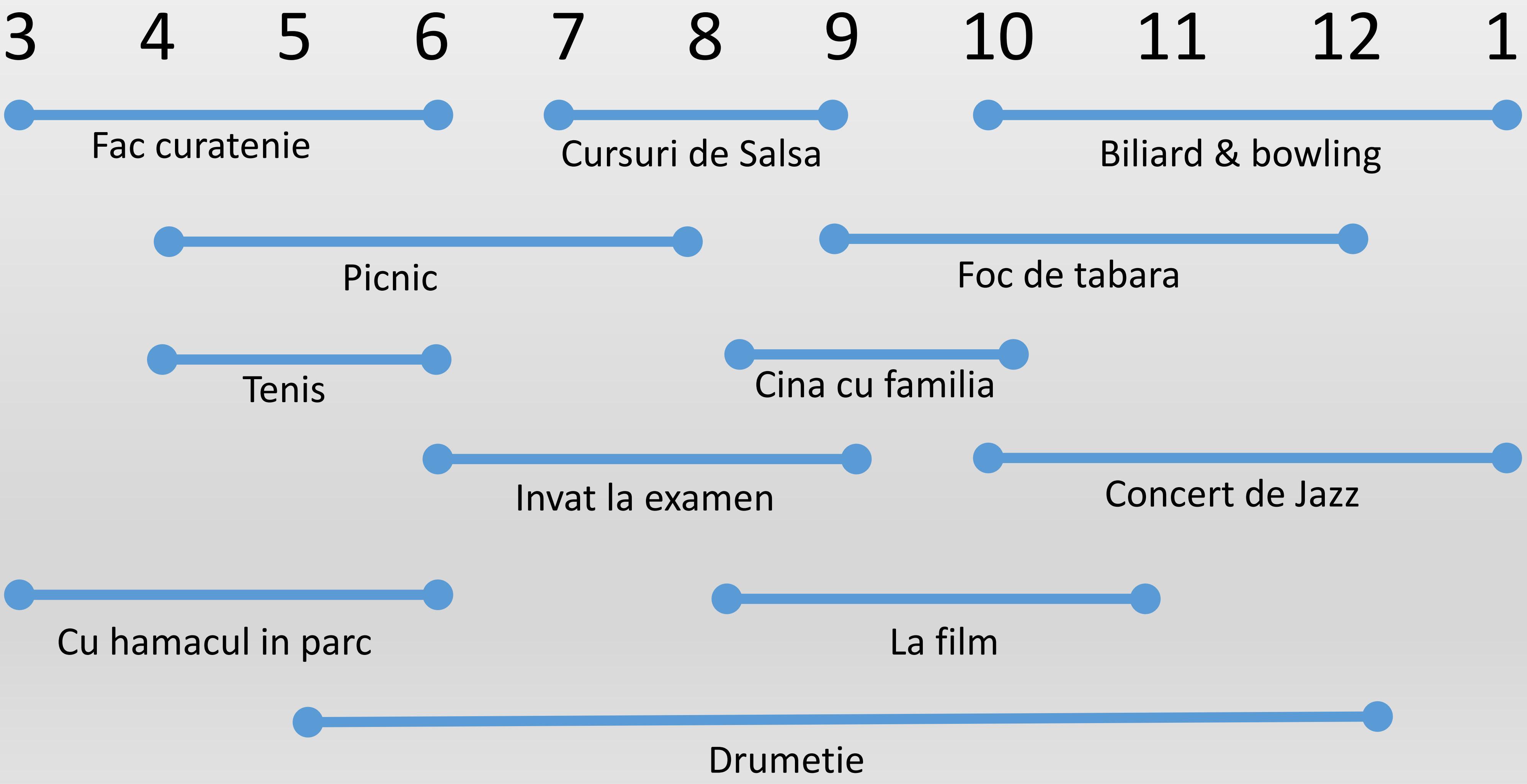
Selectia activitatilor – sa planificam o zi de weekend !

- Se da o lista de activitati $S=\{a_1, a_2, \ldots a_n\}$, fiecare avand un timp de inceput si de sfarsit $a_i = (s_i, f_i)$, $0 \leq s_i < f_i < \infty$
- Toate activitatile sunt la fel de atractive
- Dorim sa <u>maximizam</u> numarul de activitati realizate intr-o zi
- Scop: alegem numarul maxim de activitati care nu se suprapun!
  - Activitatile nu se suprapun – inseamna ca sunt *compatibile*
  - Doua activitati a_i si a_j sunt *compatibile* daca $[s_i, f_i) \cap [s_j, f_j) = \emptyset$

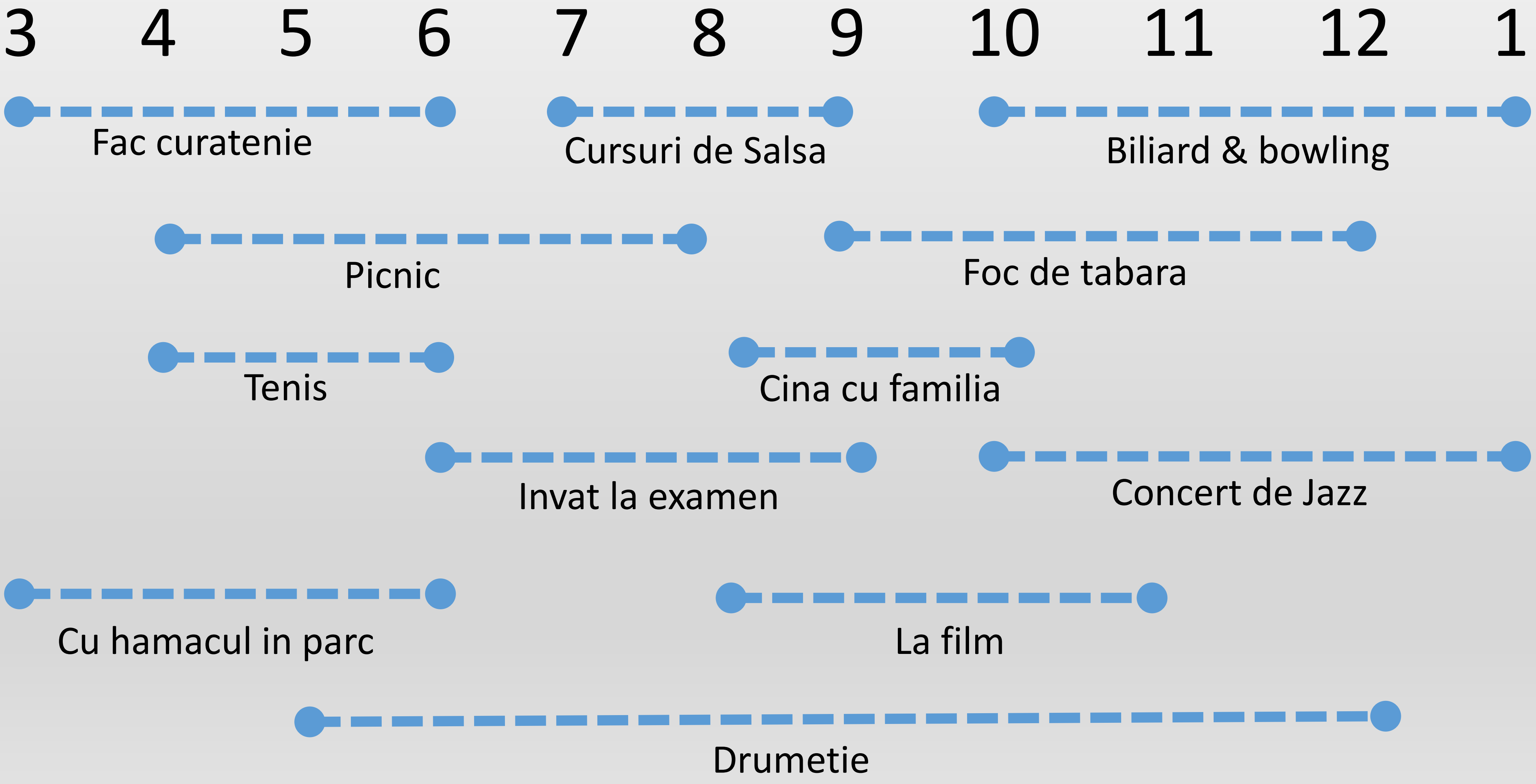
Cum gandim greedy ?

- Ca sa rezolvam problema utilizand tehnica greedy, ne gandim cum putem sa alegem activitatile care “par” cele mai avantajoase, local – euristici
- Posibile euristici?
  - In ordinea crescatoare a timpului de inceput
  - In ordinea crescatoare a duratei
  - In ordinea crescatoare a timpului de sfarsit

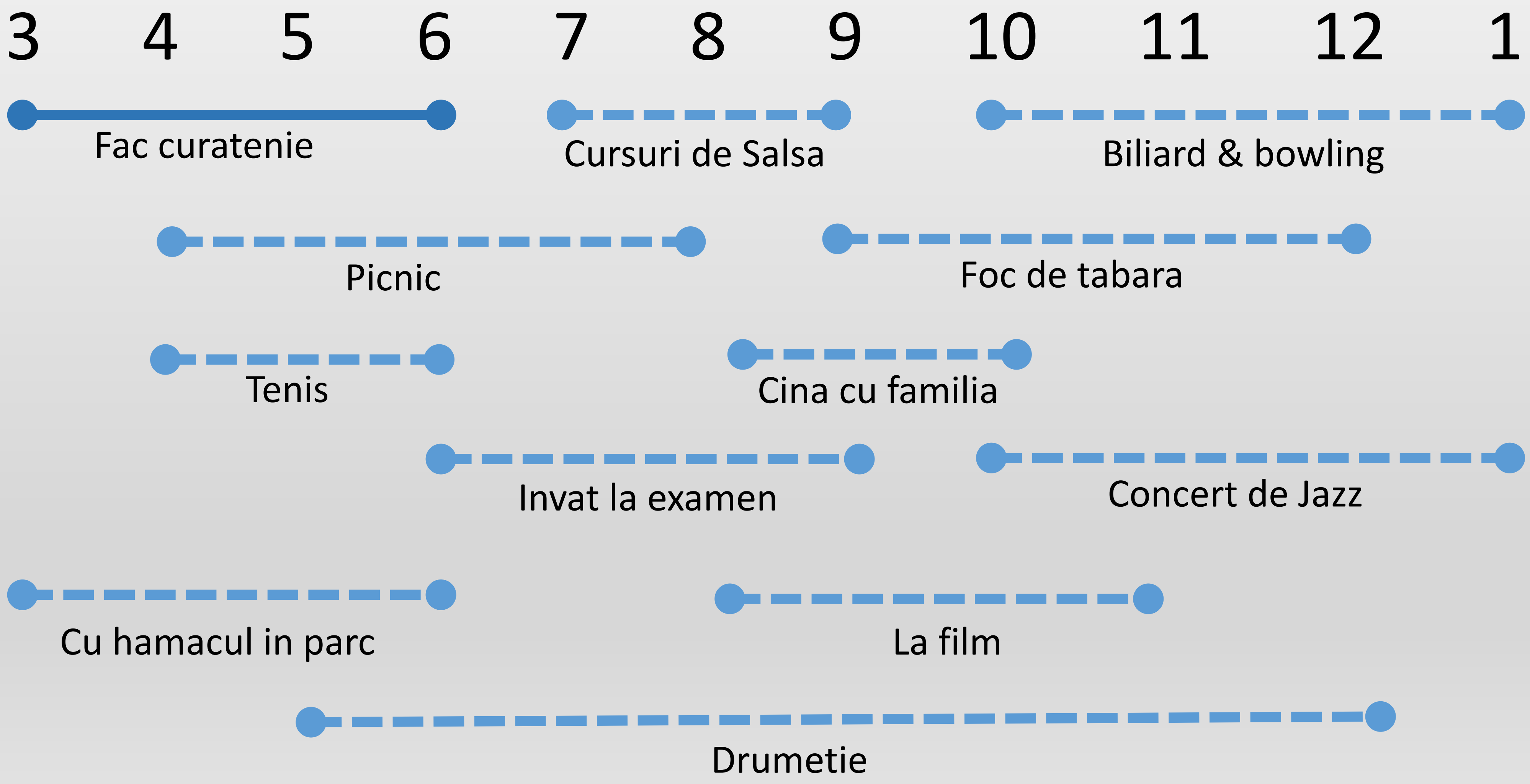
Selectia activitatilor –
In ordinea crescatoare a timpului de inceput



Selectia activitatilor –
In ordinea crescatoare a timpului de inceput

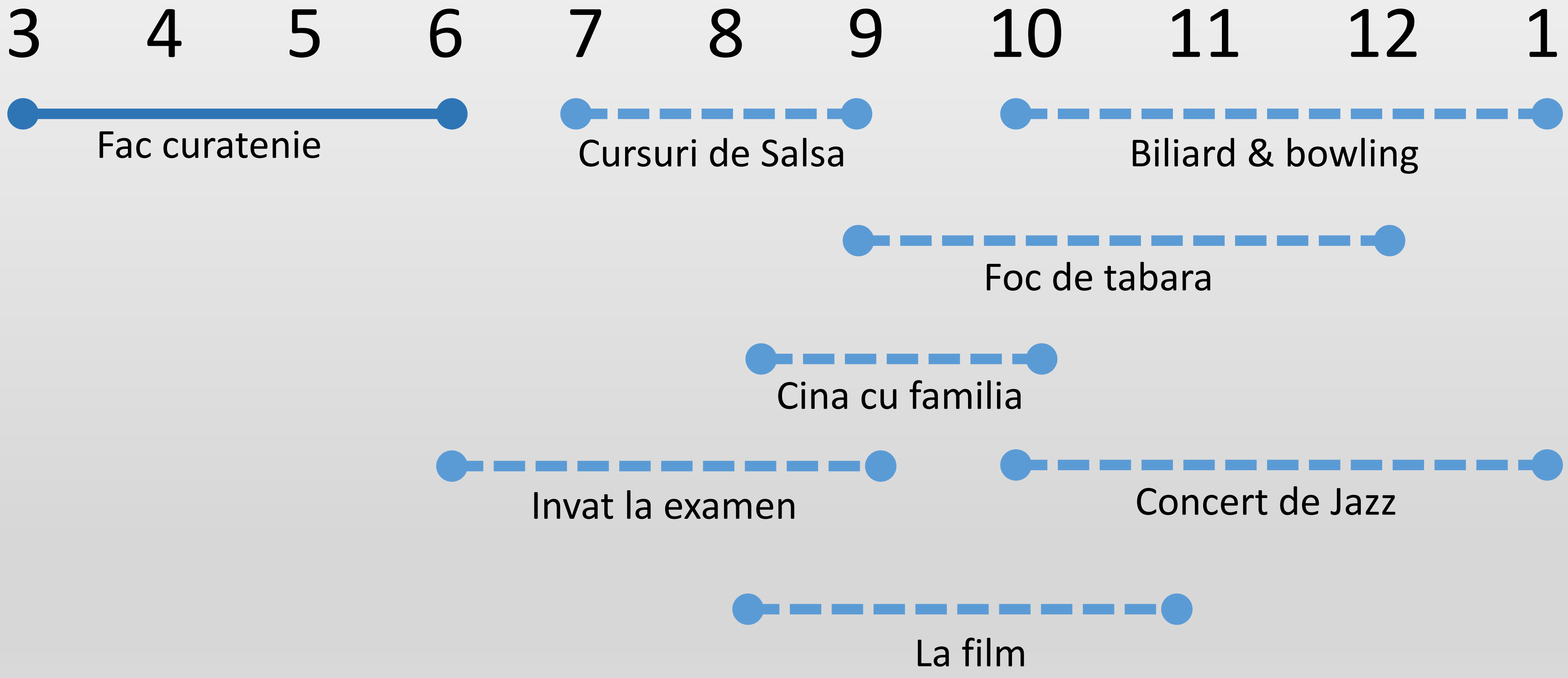


Selectia activitatilor –
In ordinea crescatoare a timpului de inceput

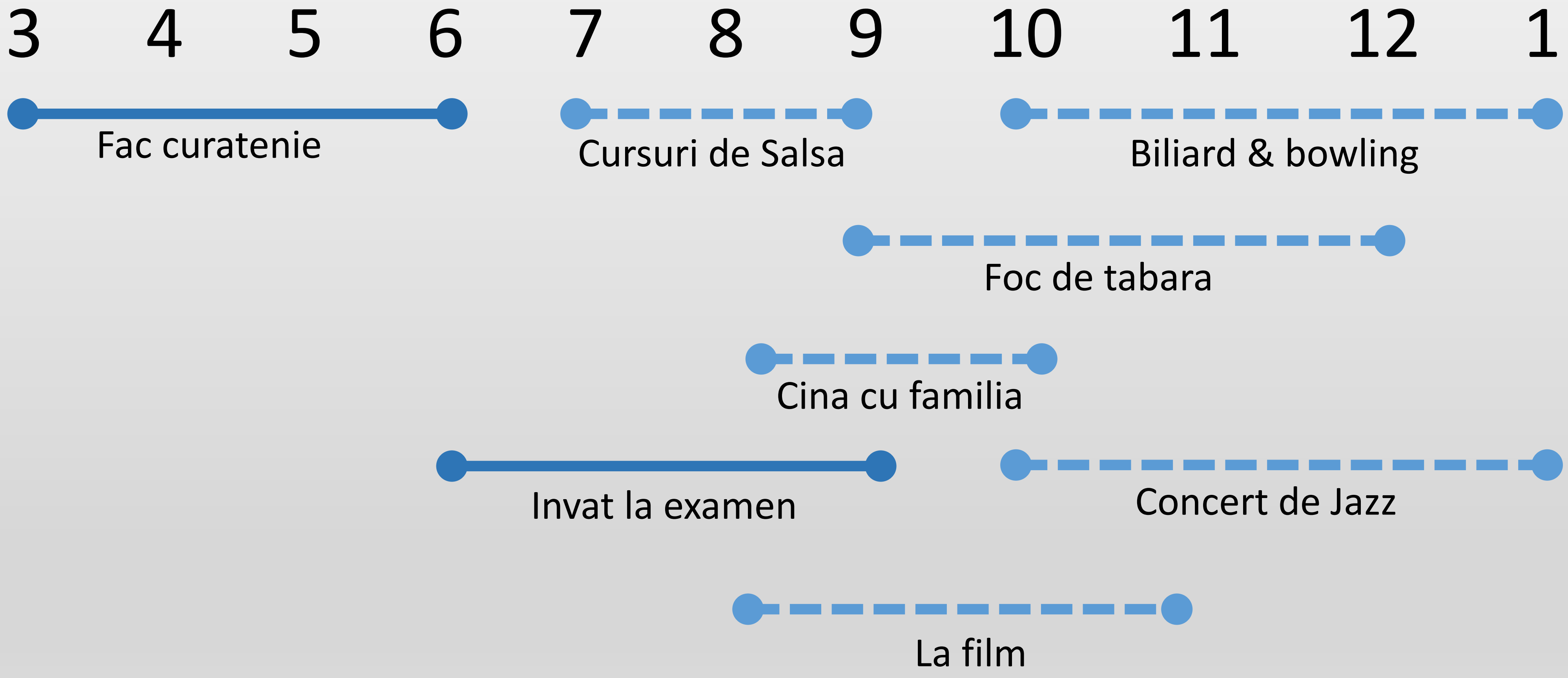


UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

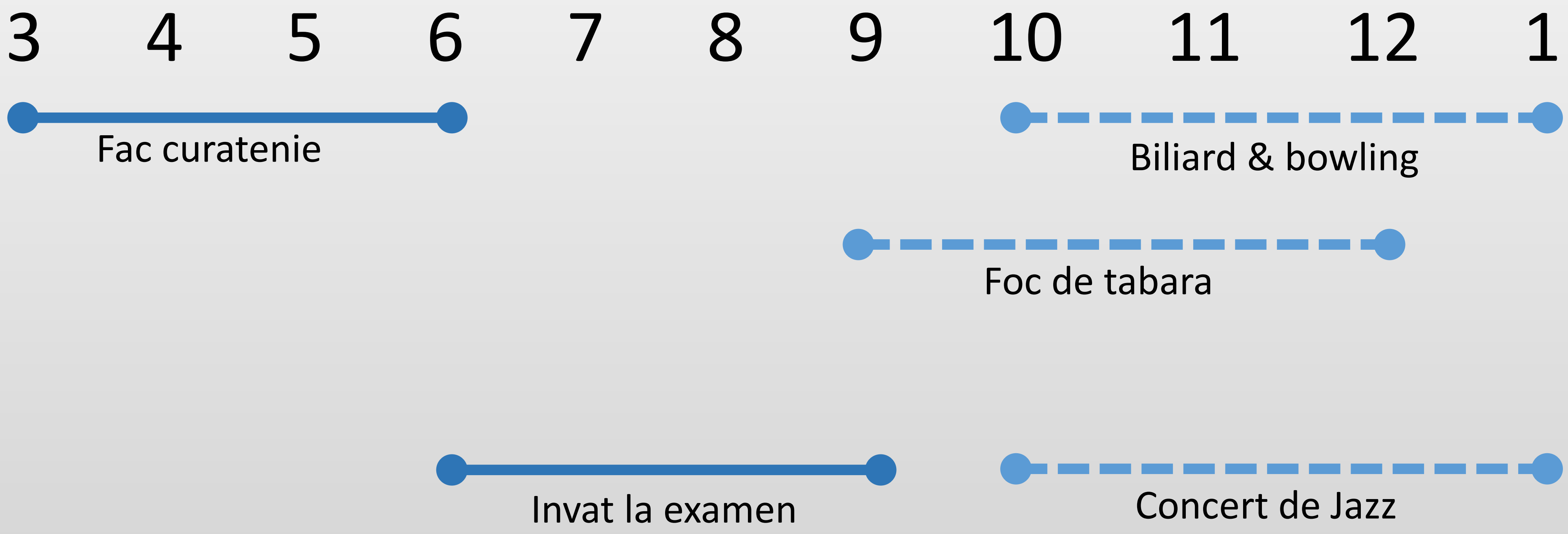
Selectia activitatilor –
In ordinea crescatoare a timpului de inceput



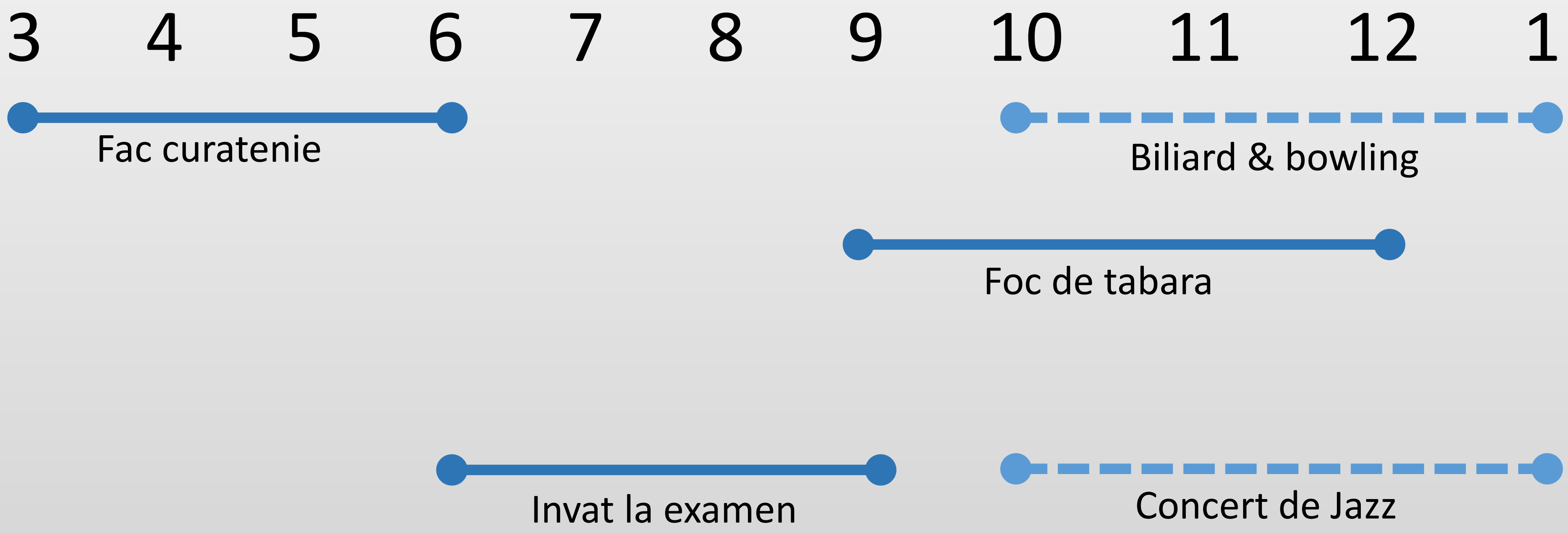
Selectia activitatilor –
In ordinea crescatoare a timpului de inceput



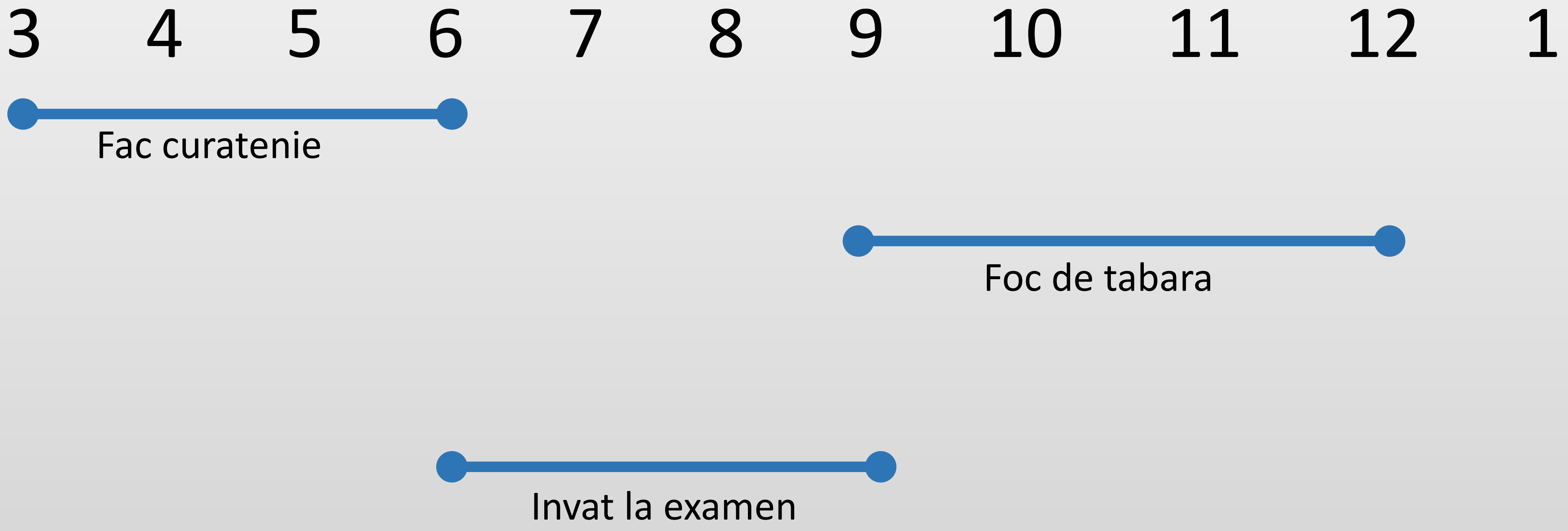
Selectia activitatilor –
In ordinea crescatoare a timpului de inceput



Selectia activitatilor –
In ordinea crescatoare a timpului de inceput



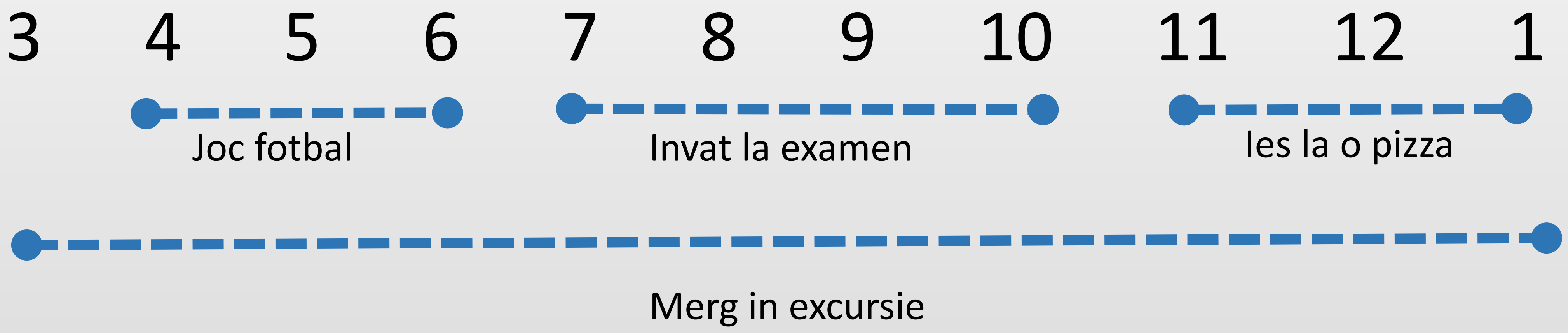
Selectia activitatilor –
In ordinea crescatoare a timpului de inceput



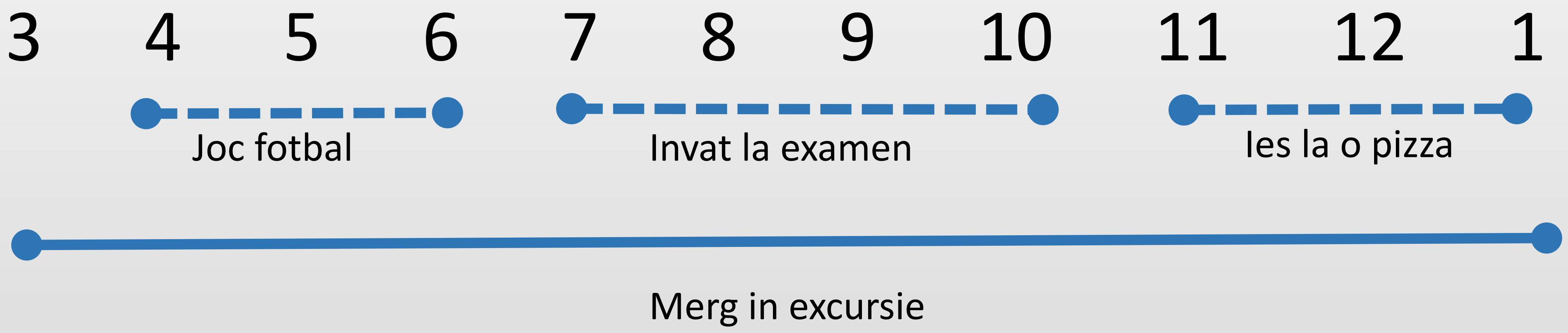
Selectia activitatilor –
In ordinea crescatoare a timpului de inceput - contraexemplu

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Selectia activitatilor – In ordinea crescatoare a timpului de inceput - contraexemplu



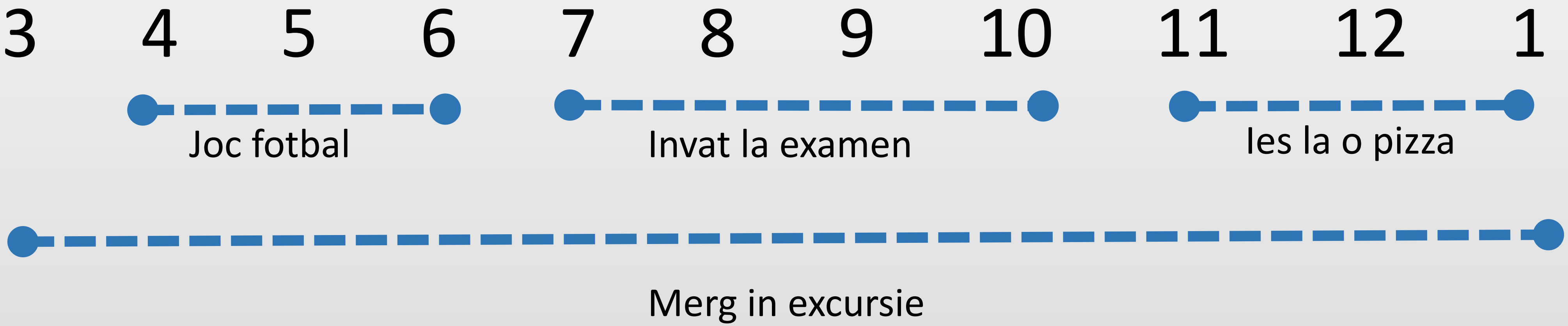
Selectia activitatilor –
In ordinea crescatoare a timpului de inceput - contraexemplu



Selectia activitatilor –
In ordinea crescatoare a timpului de inceput - contraexemplu

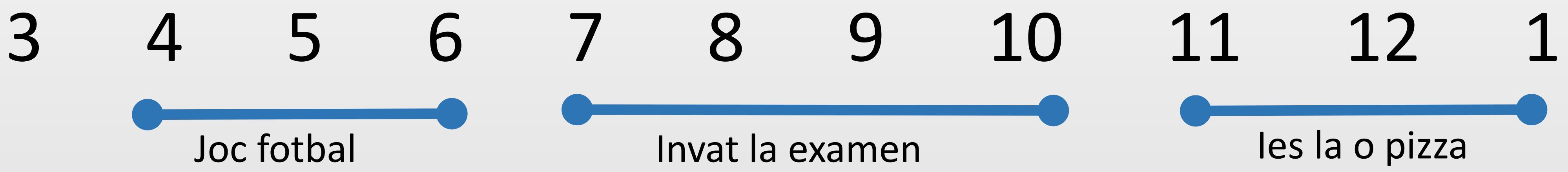
Cate activitati am ales?

Selectia activitatilor –
In ordinea crescatoare a timpului de inceput - contraexemplu



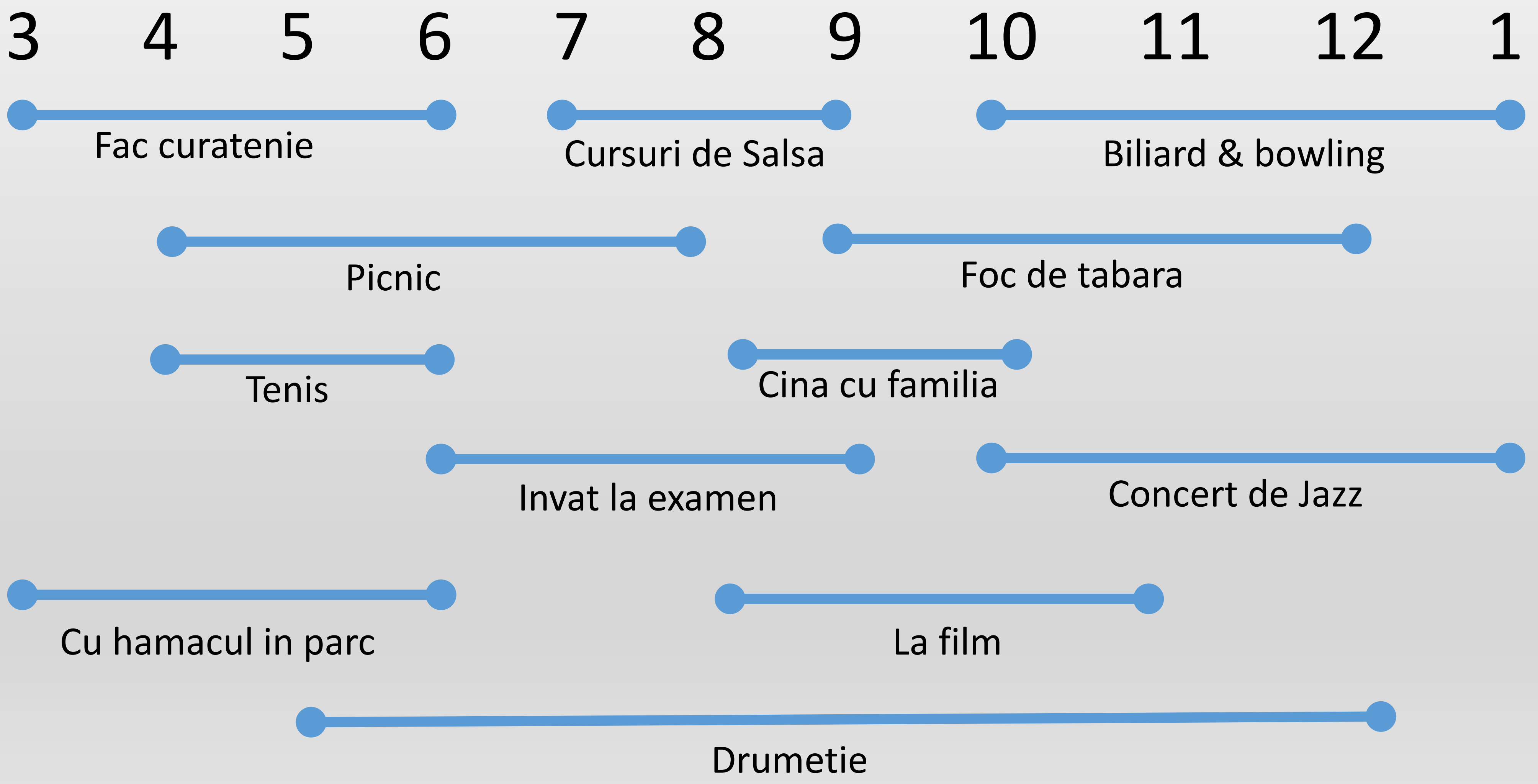
O alta alegere – mai buna ?

Selectia activitatilor –
In ordinea crescatoare a timpului de inceput - contraexemplu

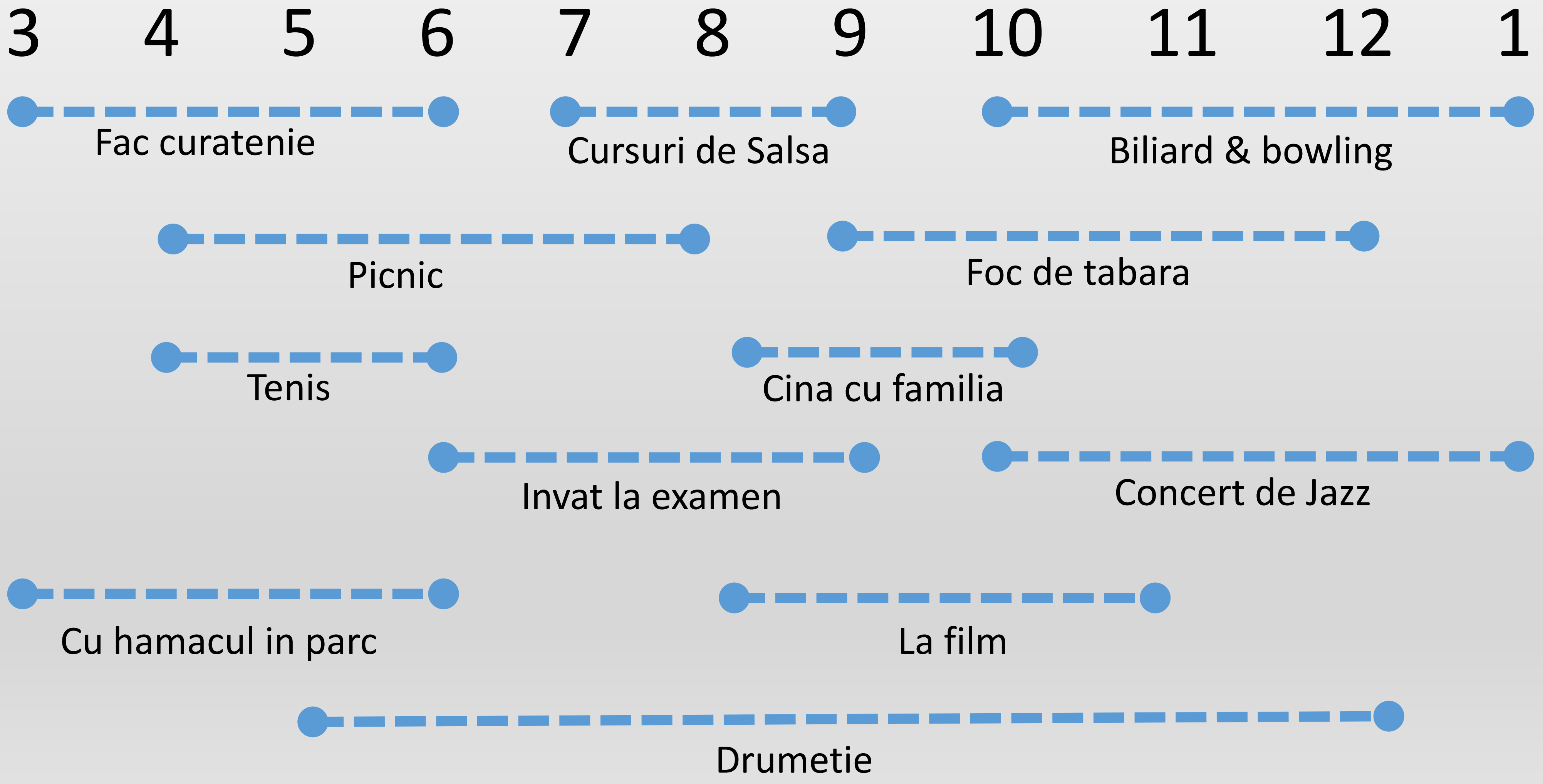


O alta alegere – mai buna ?
Cate activitati am ales ?

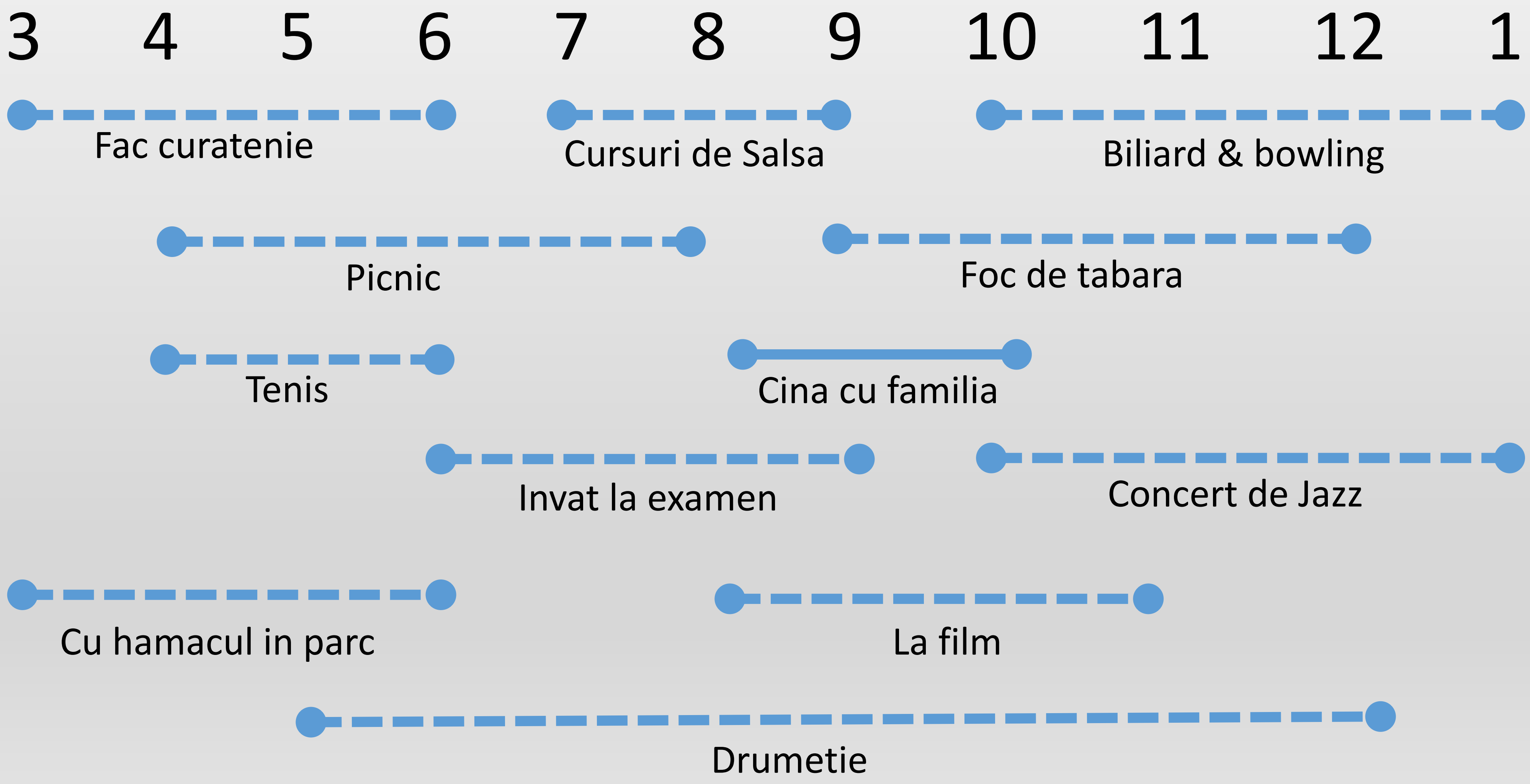
Selectia activitatilor –
In ordinea crescatoare a duratei



Selectia activitatilor –
In ordinea crescatoare a duratei

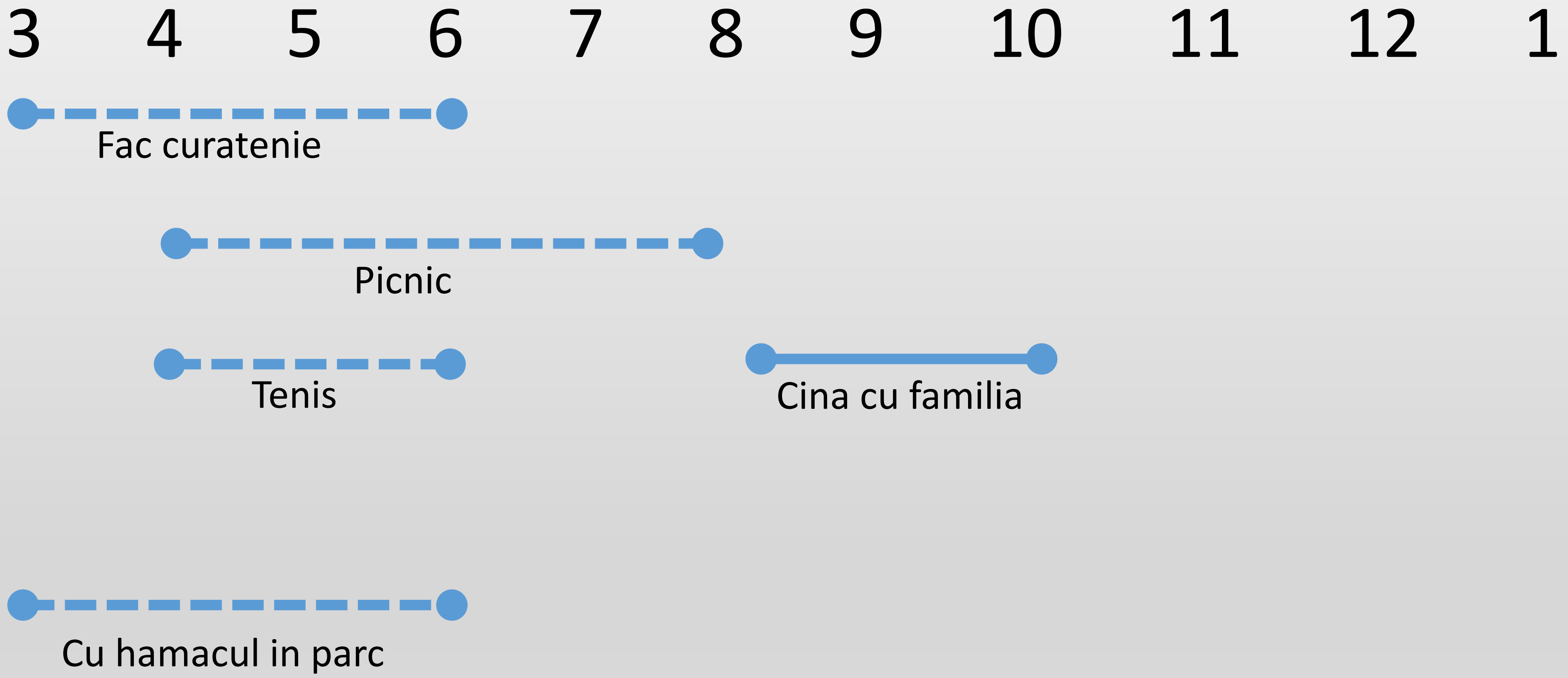


Selectia activitatilor –
In ordinea crescatoare a duratei

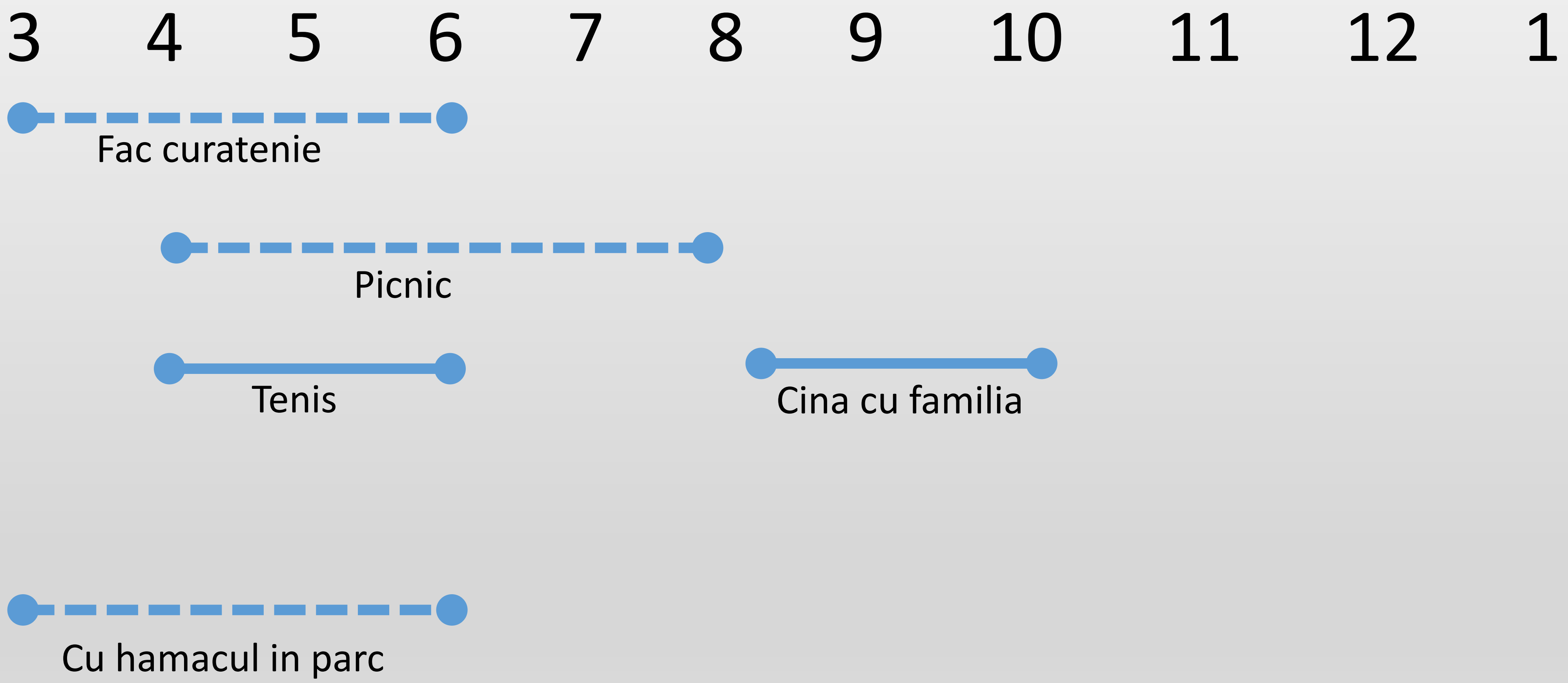


UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Selectia activitatilor – In ordinea crescatoare a duratei

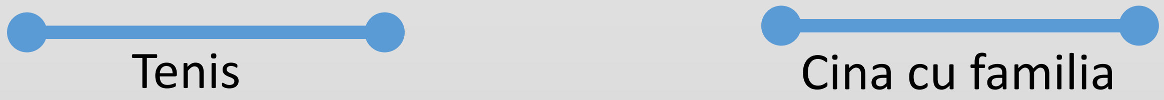


Selectia activitatilor –
In ordinea crescatoare a duratei



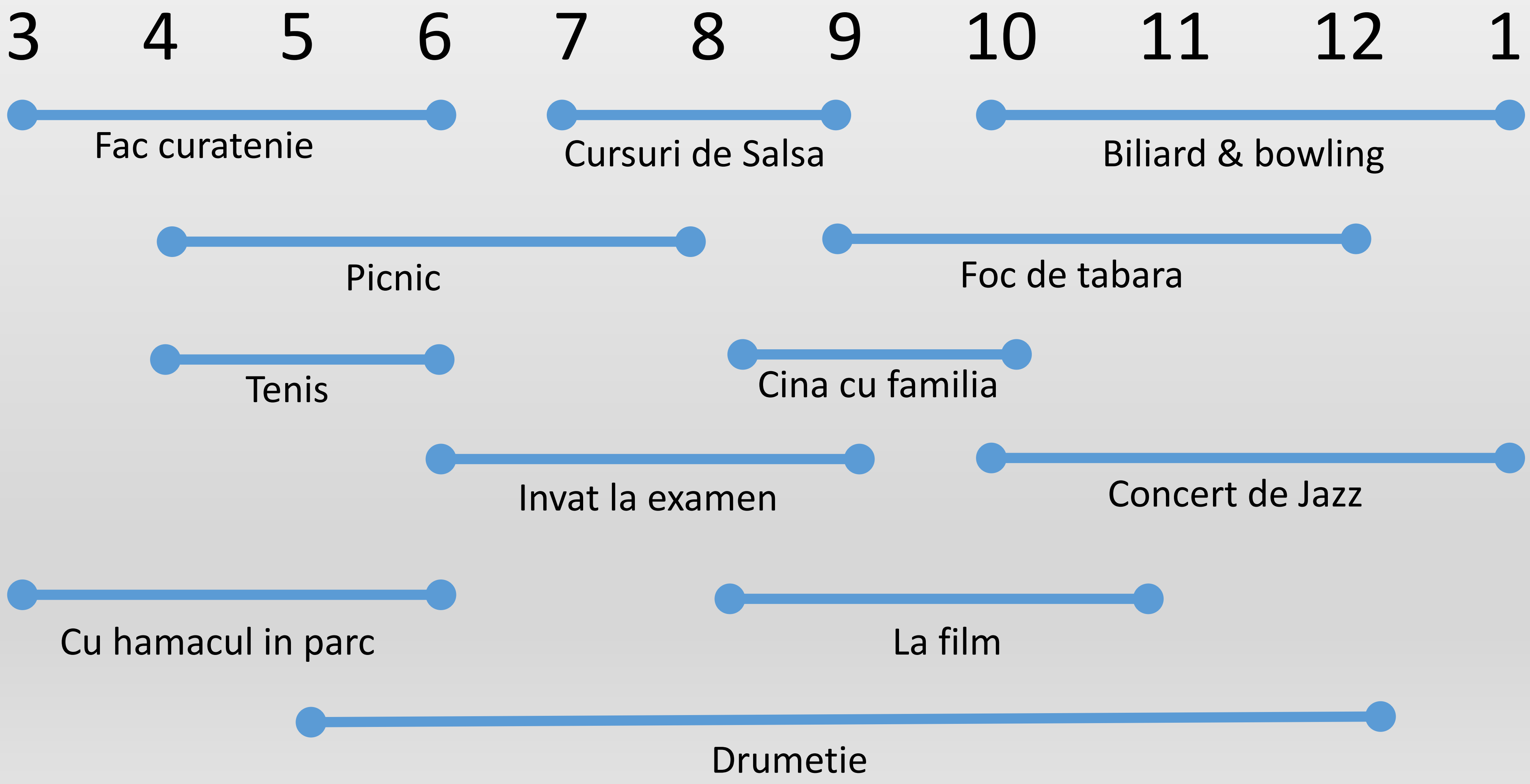
Selectia activitatilor –
In ordinea crescatoare a duratei

3 4 5 6 7 8 9 10 11 12 1

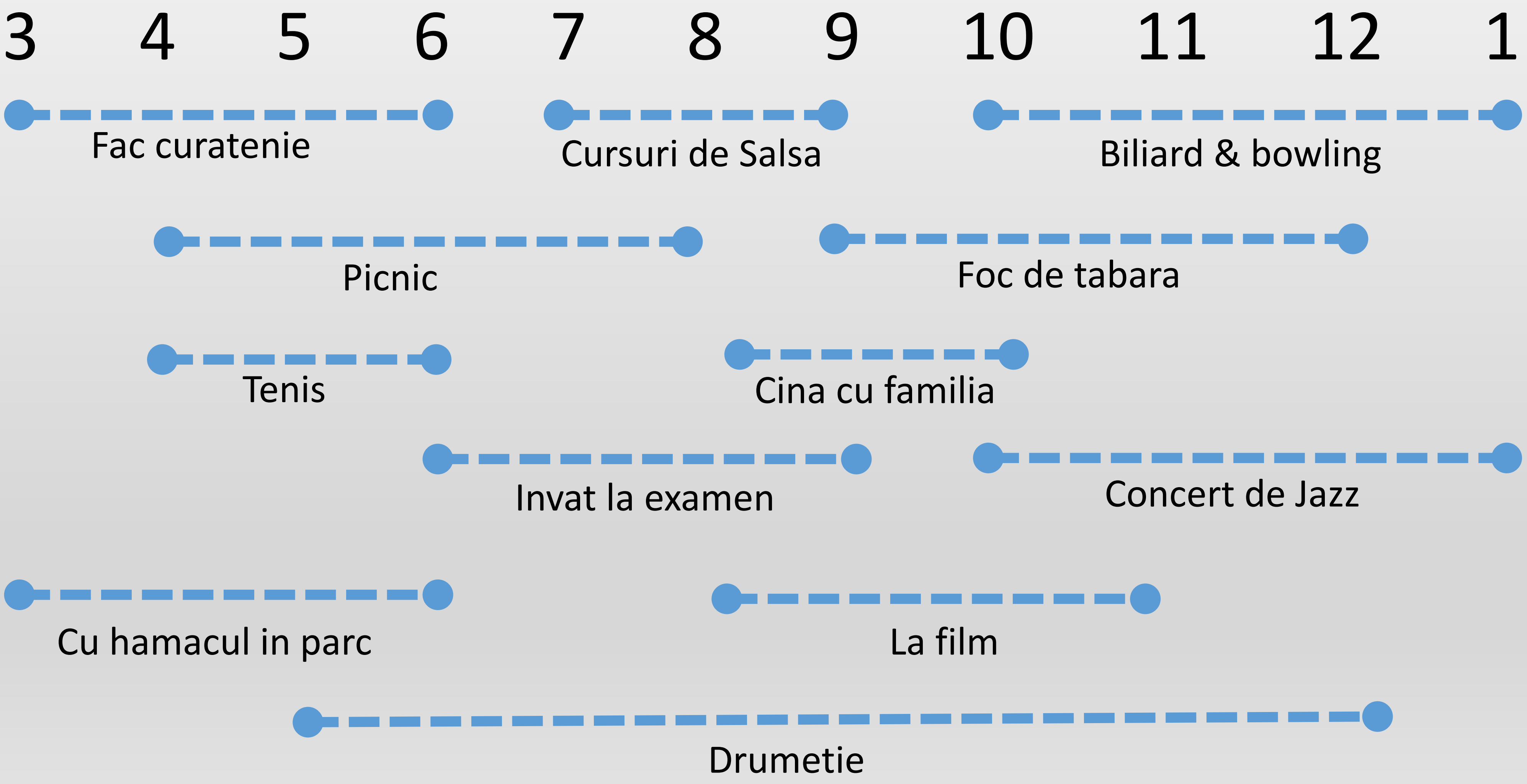


Cate activitati am ales ? Este acest numar maxim?

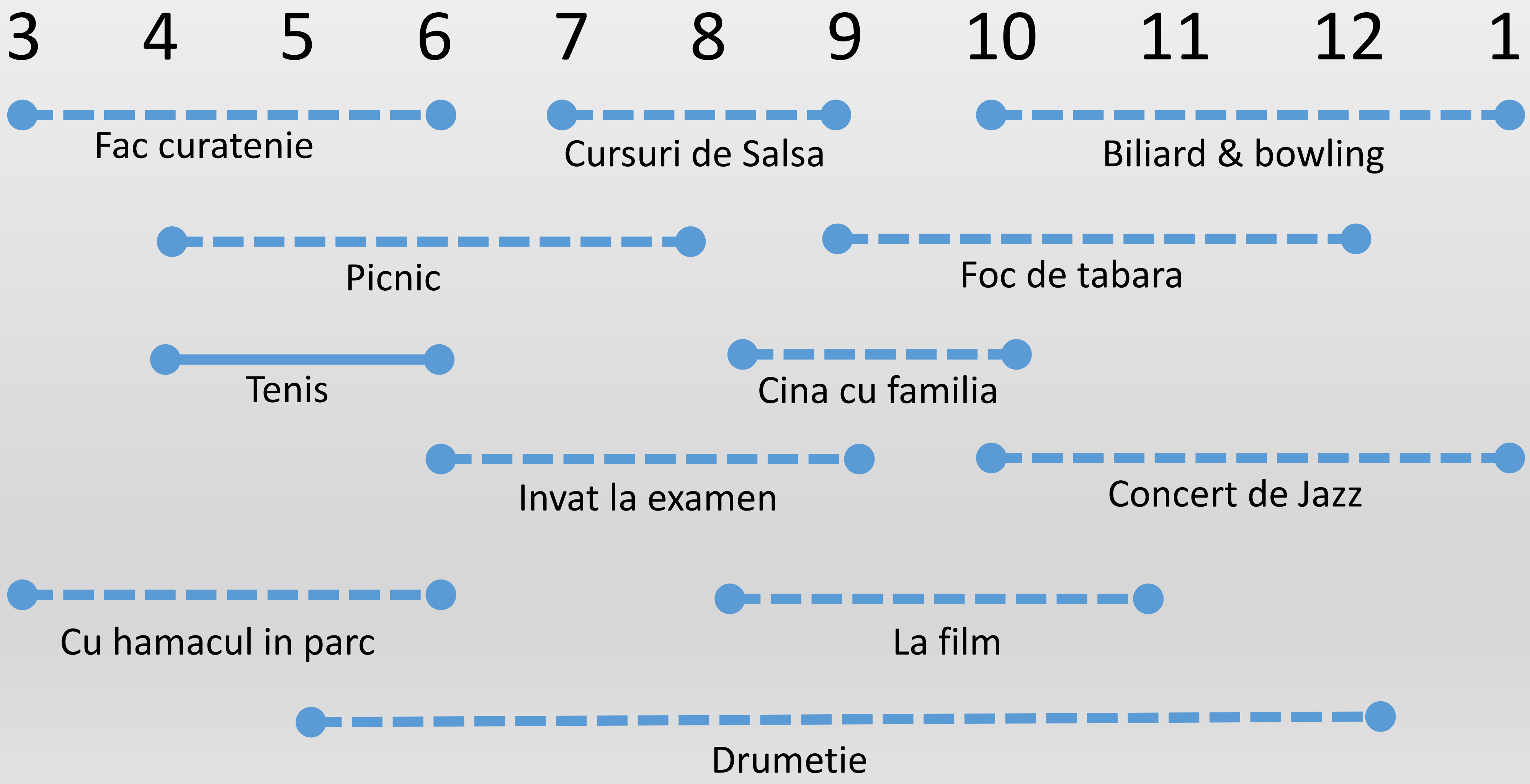
Selectia activitatilor –
In ordinea crescatoare a timpului de terminare



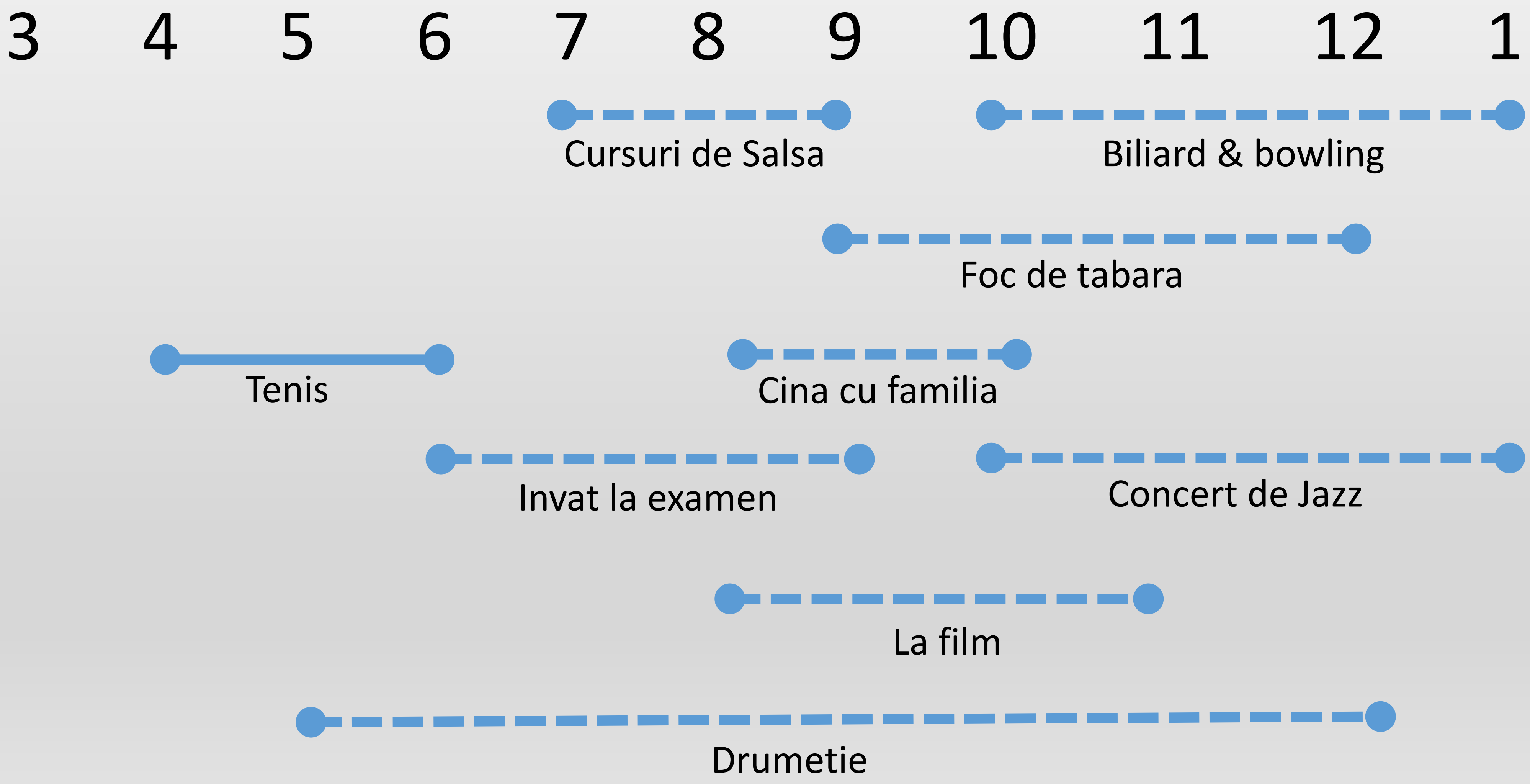
Selectia activitatilor –
In ordinea crescatoare a timpului de terminare



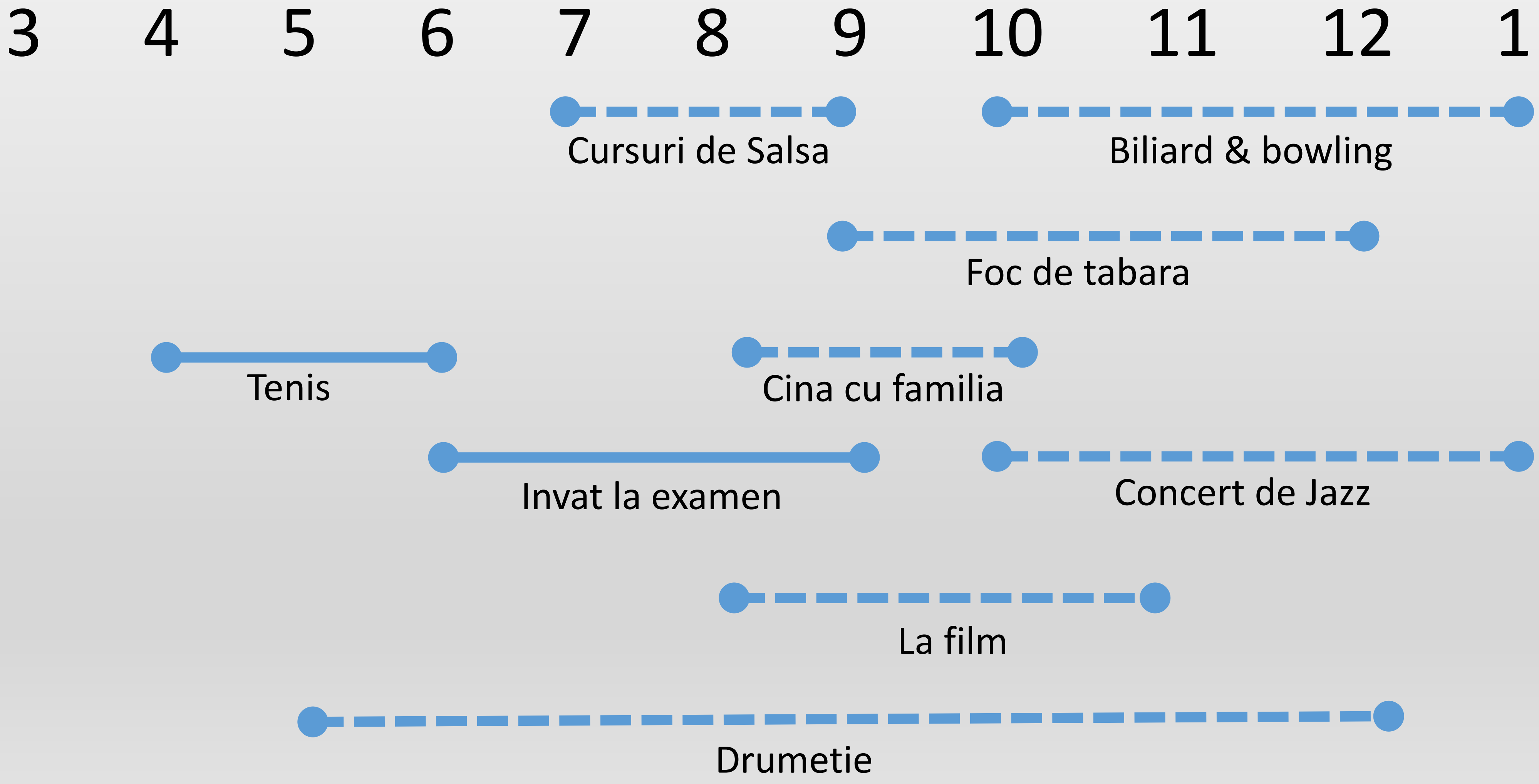
Selectia activitatilor –
In ordinea crescatoare a timpului de terminare



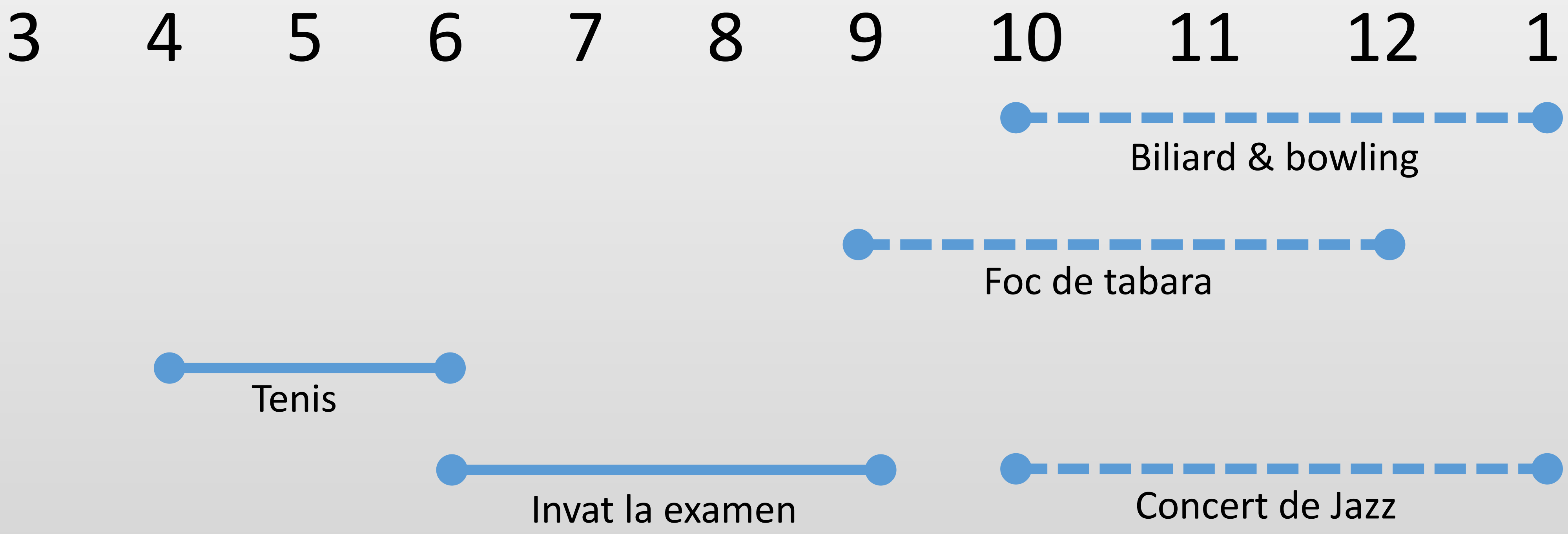
Selectia activitatilor –
In ordinea crescatoare a timpului de terminare



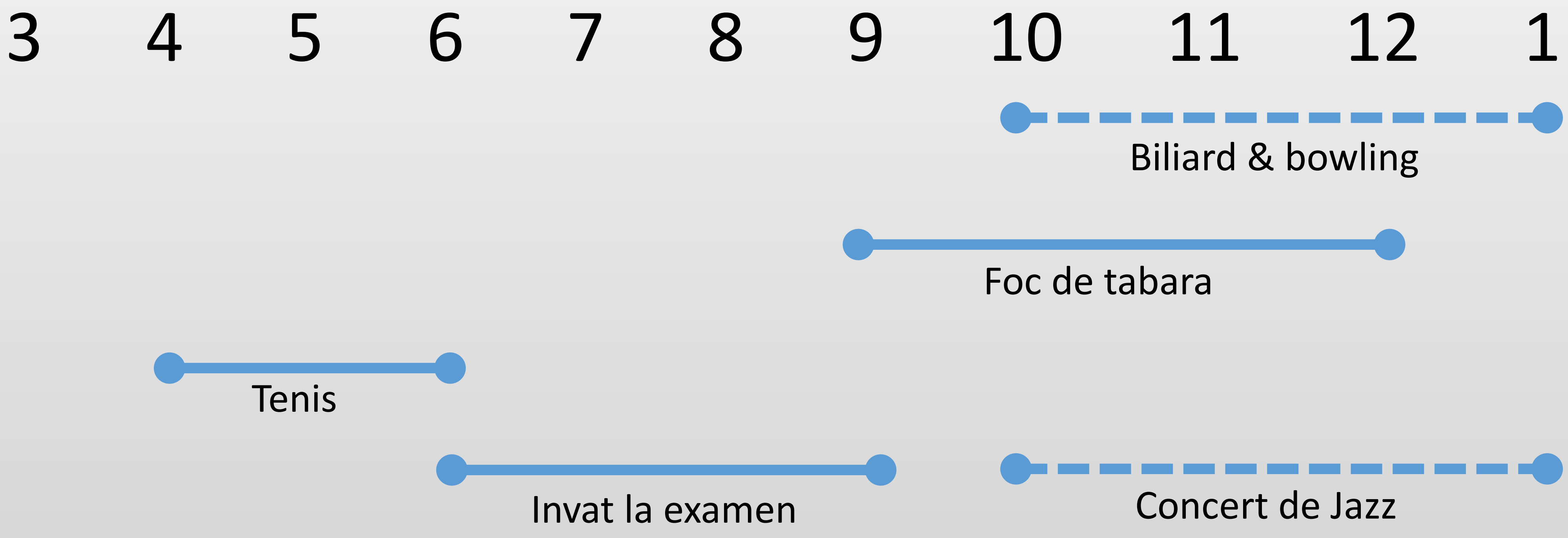
Selectia activitatilor –
In ordinea crescatoare a timpului de terminare



Selectia activitatilor –
In ordinea crescatoare a timpului de terminare

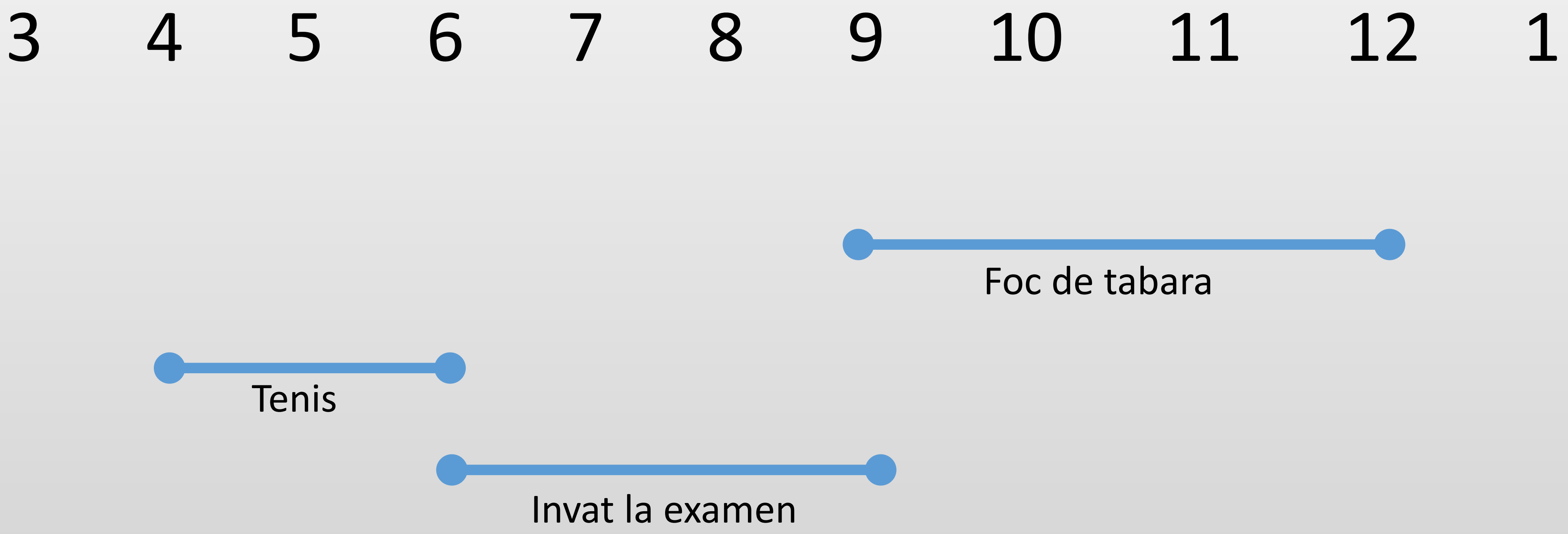


Selectia activitatilor –
In ordinea crescatoare a timpului de terminare



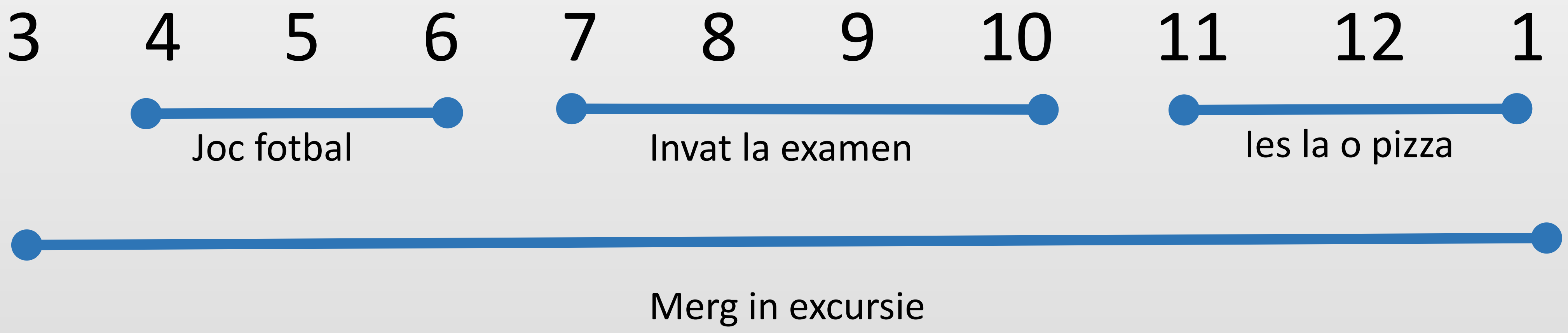
UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Selectia activitatilor –
In ordinea crescatoare a timpului de terminare

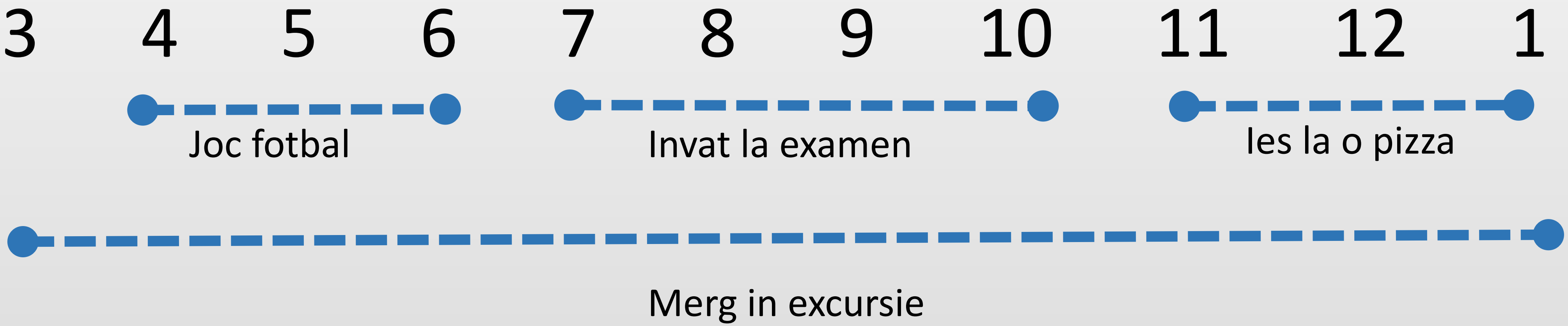


Cate activitati am ales ? Este acest numar maxim?

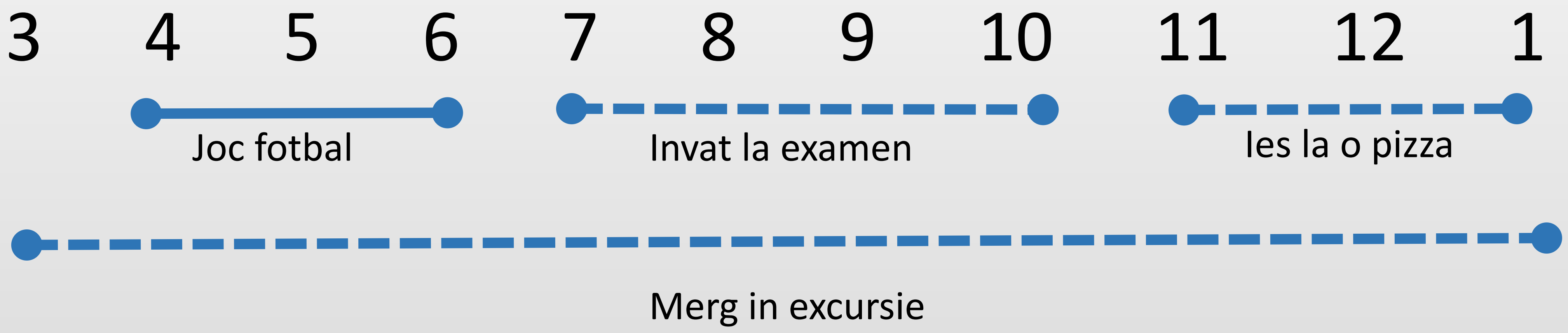
Selectia activitatilor –
In ordinea crescatoare a **timpului de terminare**



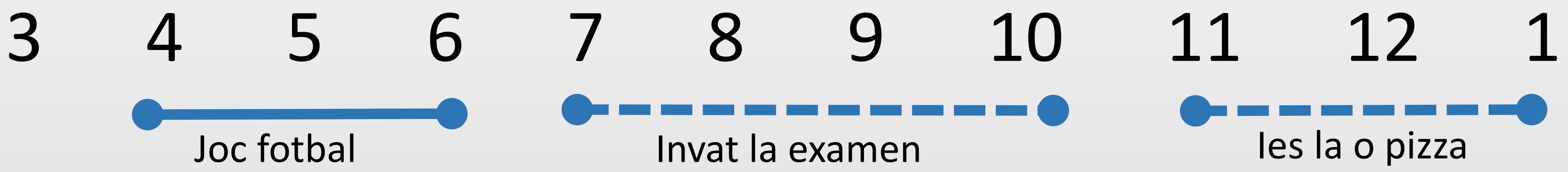
Selectia activitatilor –
In ordinea crescatoare a **timpului de terminare**



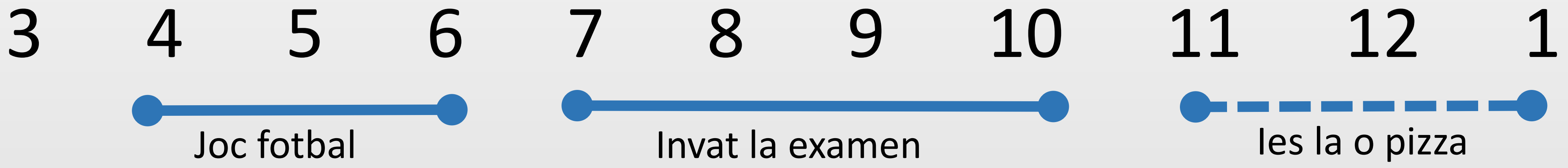
Selectia activitatilor –
In ordinea crescatoare a **timpului de terminare**



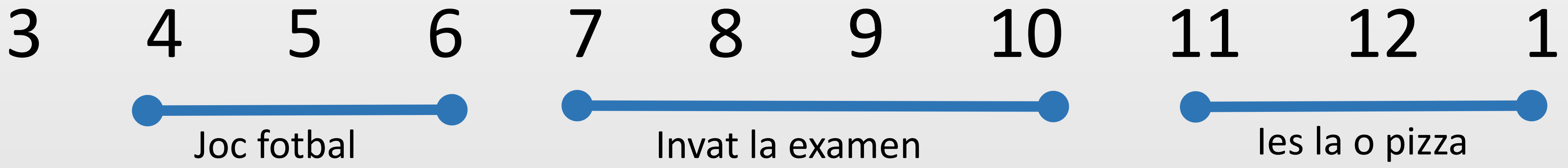
Selectia activitatilor –
In ordinea crescatoare a **timpului de terminare**



Selectia activitatilor –
In ordinea crescatoare a **timpului de terminare**



Selectia activitatilor – In ordinea crescatoare a **timpului de terminare**



Tehnica greedy: Selectia activitatilor

- Am vazut ca functioneaza selectia in functie de timpul de finalizare. Puteti oferi un contraexemplu pentru aceasta euristica?
- $A = \{a_1, a_2, \ldots a_n\}$, cu $a_i=(s_i, f_i)$, S este solutia

Greedy-activity-selector(A)
Ordonam crescator activitatile din A in ordinea crescatoare a timpului de finalizare, f_i
$S \leftarrow \emptyset$

Cat timp *A* nu e vida
  Alegem o activitate a_k cu timpul cel mai mic de finalizare
  Adaugam activitatea la multimea solutiilor, $S=S \cup \{ a_k\}$
  Stergem din A activitatile care se suprapun cu a_k

Complexitate: ?

Greedy-activity-selector(A)
Sort activities in ascending order of finish time
N = A.length
$S \leftarrow \{a_1\}$
k = 1
for m = 2 to n do
  if $(s_m \geq f_k)$ then
    $S = S \cup \{a_m\}$
    $k = m$
Return S

Demonstratie – greedy alege solutia optima

Teorema – pentru problema selectiei activitatilor metoda greedy alege solutia optima. Inainte de a demonstra acest lucru, consideram urmatoarele:

Observatie:

Activitatea k aleasa de metoda greedy se termina mai repede / devreme decat activitatea k aleasa de orice alta solutie care reprezinta o combinare corecta a activitatilor.

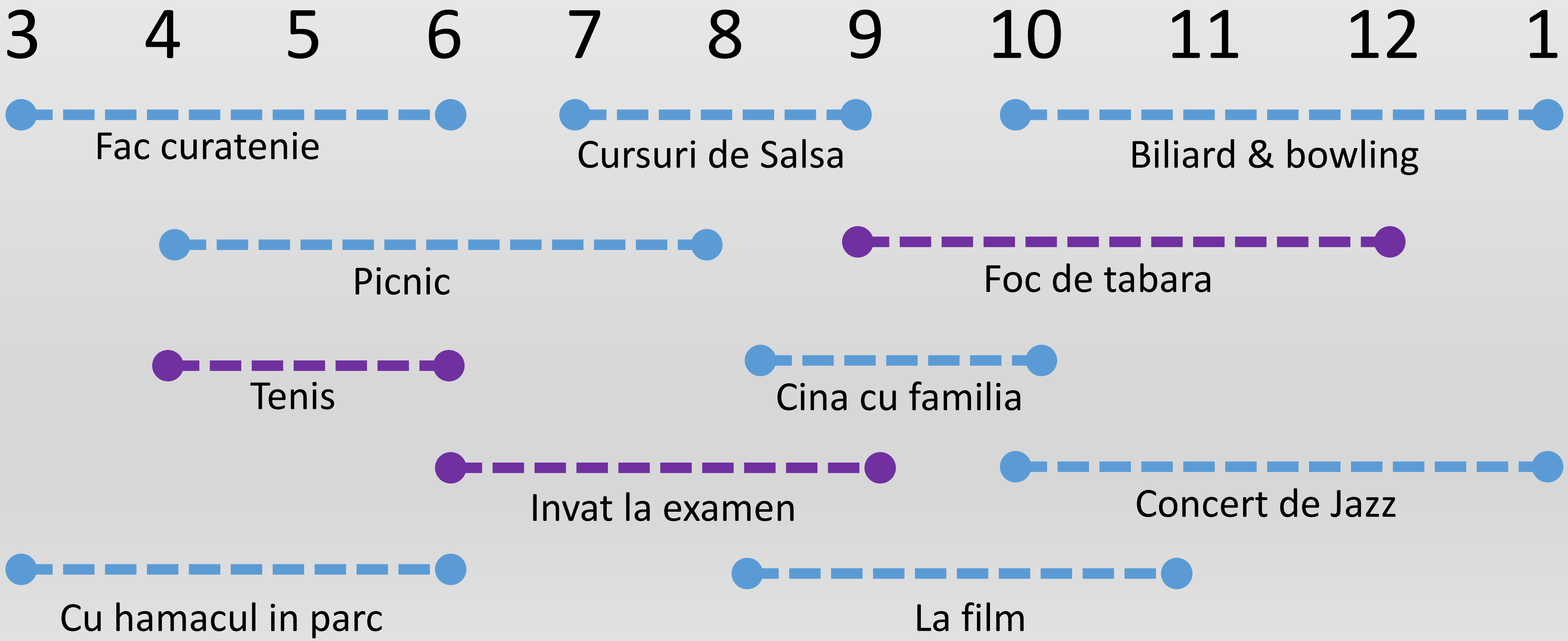
E nevoie sa demonstram ca acest lucru este adevarat si sa aratam ca daca e adevarat, algoritmul greedy genereaza solutia optima.

- Notam cu $f(i, S)$ timpul de terminare a activitatii i in solutia S

Demonstratie – greedy alege solutia optima

Observatie:

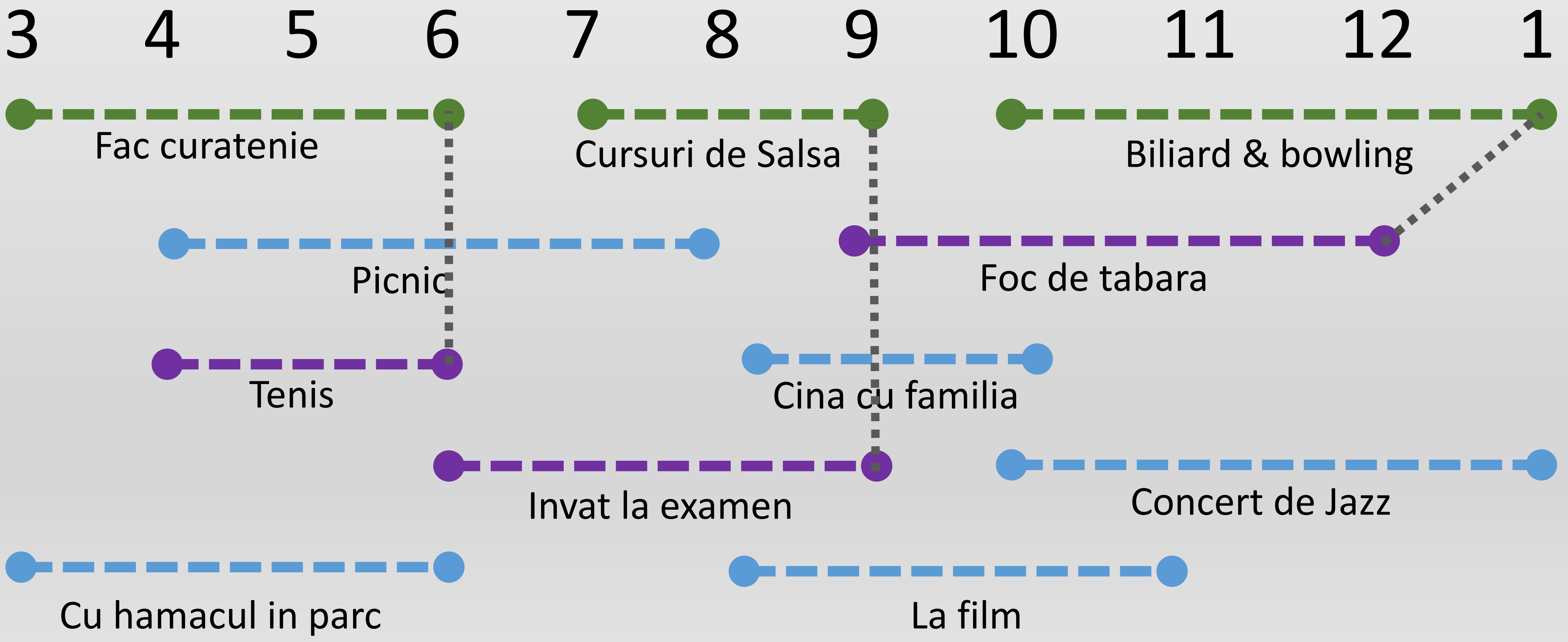
Activitatea k aleasa de metoda greedy se termina mai repede / devreme decat activitatea k aleasa de orice alta solutie care reprezinta o combinare corecta a activitatilor.



Demonstratie – greedy alege solutia optima

Observatie:

Activitatea k aleasa de metoda greedy se termina mai repede / devreme decat activitatea k aleasa de orice alta solutie care reprezinta o combinare corecta a activitatilor.



Demonstratie – greedy alege solutia optima

Lema – Daca *S e o solutie data de greedy si S* e solutia optima*, atunci pentru oricare $1 \le i \le |S|$, avem $f(i, S) \le f(i, S^*)$.

Demonstratie prin inductie:

1. Prima activitate are f(1, S) ≤ f(1, S*) pentru ca asa fost selectata – are timpul minim de terminare.
2. Inductia: presupunem ca pentru activitatea i cu 1 ≤ i < |S| este adevarata proprietatea f(i, S) ≤ f(i, S*), activitatea i din solutia S se termina inaintea activitatii i din solutia S*
3. Demonstram pentru i+1: in Solutia S* activitatea (i+1) incepe dupa ce se termina activitatea i din S*, folosind punctul 2) este adevarat ca activitatea i+1 din S* incepe dupa ce se termina activitatea i din S.
4. Astfel activitatea (i+1) din S* trebuie sa fie in multimea activitatilor A cand algoritmul greedy face selectia activitatii (i+1) din S. Cum greedy selecteaza activitatile din A in ordinea crescatoare a timpului de finalizare, avem ca f(i + 1, S) ≤ f(i + 1, S*)

Demonstratie – greedy alege solutia optima

Teorema – pentru problema selectiei activitatilor metoda greedy alege solutia optima.

Demonstratie:

- Fie S solutia determinata de metoda greedy si fie S* o solutie optima.
- S* este optima deci $|S| \le |S^*|$
- Vom demonstra ca $|S| \ge |S^*|$

Demonstratie – greedy alege solutia optima

- Presupunem prin contradictie ca $|S| < |S^*|$.
- Fie $k = |S|$.
- Folosind lema din slide-ul anterior stim ca $f(k, S) \leq f(k, S^*)$, deci activitatea k din S se termina mai repede decat activitatea k din S*
- Dar s-a presupus ca $|S| < |S^*|$ deci exista in S* activitatea $(k + 1)$ iar timpul ei de start este dupa timpul de terminare a activitatii k din S*, adica $f(k, S^*)$, respectiv dupa $f(k, S)$.
- Deci dupa ce algoritmul greedy a adaugat activitatea k la Solutia S, in multimea de activitati A ar mai exista activitatea k+1 din S*. Dar algoritmul greedy se termina cand nu mai poate adauga activitati deci A este vida -> contradictie -> presupunerea este falsa deci $|S| \geq |S^*|$.
- Stim ca $|S| \leq |S^*|$, am demonstrat ca $|S| \geq |S^*|$ deci $|S| = |S^*|$

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Problema numararii restului

- *<u>Problema</u>*: Un vanzator are la dispozitie o colectie de monede si bancnote de diferite valori. Se cere sa formeze o suma <u>specificata</u> folosind <u>numarul minim de bancnote</u>.

Problema numararii restului

Formulare matematica:

- Se dau ***n*** bancnote si monede: $\boldsymbol{P=\{p_1, p_2, \ldots, p_n\}}$
- putem avea repetitii (2 bancnote de 1 leu, etc)
- fie $\boldsymbol{d_i}$ valoarea lui $\boldsymbol{p_i}$
- Gasiti cea mai mica submultime S a lui P, S ⊆ P, astfel incat

$$\sum_{p_i \in S} d_i = A$$

unde A este suma de returnat.

Problema numararii restului

- In practica un vanzator nu considera toate posibilitatile de numarare a sumei date
- In schimb, numara incepand de la cea mai mare valoare si trecand apoi la valori mai mici
- Odata ce o bancnota a fost selectata, nu mai sunt considerate solutii fara acea bancnota
- Daca bancnotele/monezile sunt gata sortate - timpul de rulare a acestui algoritm este O(n).
- **Nu garanteaza gasirea solutiei optime!**

Exemplu:

$\{d_1, d_2, \ldots, d_{10}\} = \{\mathbf{5, 5, 10, 10, 10, 10, 10, 50, 200, 200, 200, 500}\}$.

Suma 520 lei: 500 + 10 + 10

Dar suma 600 lei ?

Solutia care pleaca de la ordinea descrescatoare
A bancnotelor:

DAR Solutia cu numar minim de bancnote este:

Problema numararii restului

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

La fiecare iteratie se adauga bancnota cu cea mai mare valoare, avand proprietatea ca suma obtinuta dupa adaugarea bancnotei sa nu fie mai mare decat suma de platit (A).

- X – suma de platit
- c_1, c_2, … c_n ,sunt valorile monedelor si bancnotelor
- Complexitate ?

```
CASHIERS-ALGORITHM (x, c1, c2, …, cn)
-----------------------------------------------------------
SORT n coin denominations so that c1 < c2 < … < cn
S ← φ          ←—— set of coins selected
WHILE x > 0
   k ← largest coin denomination ck such that ck ≤ x
   IF no such k, RETURN "no solution"
   ELSE
      x ← x − ck
      S ← S ∪ { k }
RETURN S
-----------------------------------------------------------
```

Tehnica greedy: Problema rucsacului

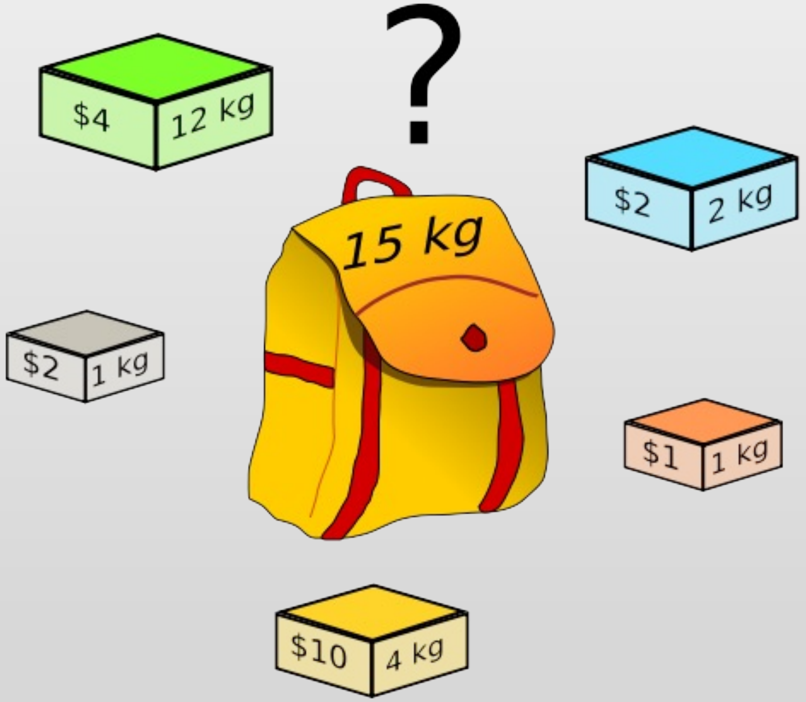
Se da:

- O multime de *n* obiecte, fiecare avand o anumita *greutate* si o anumita *valoare (profit)*
- Un rucsac de o anumita capacitate (poate contine o anumita greutate)

Se cere: sa se selecteze sub-multimea de obiecte care incape in rucsac si maximizeaza *valoarea* totala (0-1 knapsack)

Modelare:

- w_i *greutatea* obiectului ***i***,
- p_i *profitul* elementului ***i***
- ***W*** *capacitatea* rucsacului
- x_i variabila din vectorul solutie, care ia valoarea 1 cand obiectul ***i*** este carat in rucsac, si 0 altfel



Tehnica greedy: Problema rucsacului

- Dandu-se multimile $\{w_1, w_2, \ldots, w_n\}$ *si* $\{p_1, p_2, \ldots, p_n\}$, obiectivul este maximizarea: $$\sum_{i=1}^{n} p_i x_i$$

supusa constrangerii: $$\sum_{i=1}^{n} w_i x_i \leq W$$

- Seamana aceasta problema cu problema *numararii restului*?
- Solutii posibile - bazate pe *backtracking,* sau *programare dinamica (exercitiu)*

Tehnica greedy: Problema rucsacului

- Euristici posibile pentru tehnica greedy:
  - ***Greedy dupa profit***
    - la fiecare pas se selecteaza obiectul cu profit maxim
    - selecteaza cele mai profitabile obiecte intai
  - ***Greedy dupa greutate***
    - la fiecare pas se selecteaza obiectul de greutate minima
    - incearca sa maximizeze profitul maximizand numarul de obiecte din rucsac
  - ***Greedy dupa densitatea de profit***
    - la fiecare pas se selecteaza obiectul care area cea mai mare densitate de profit p_i/w_i
    - incearca sa maximizeze profitul prin selectia obiectelor cu cel mai mare profit per unitate de greutate

Tehnica greedy: Problema rucsacului

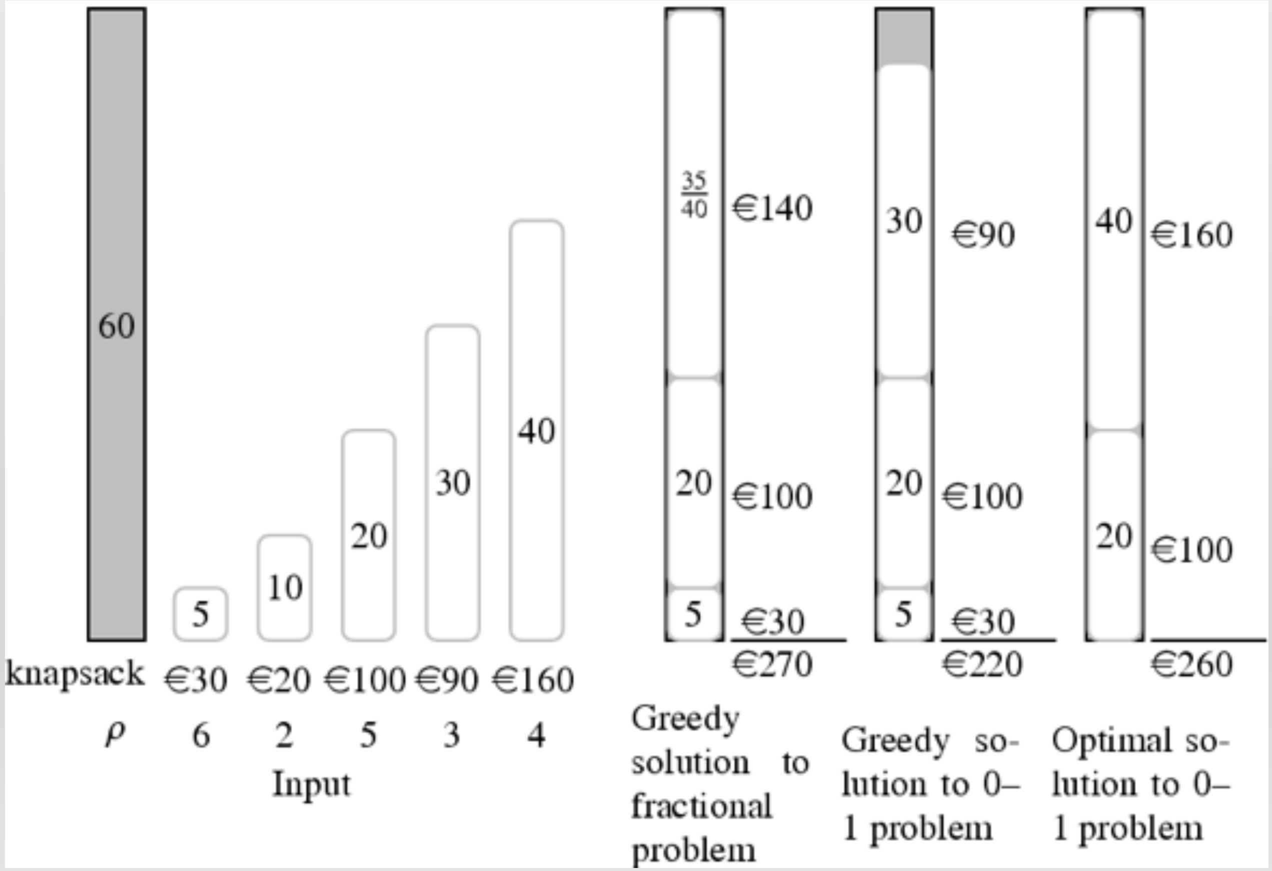
- $W = 100$

				greedy by			
i	w_i	p_i	p_i/w_i	profit	weight	density	optimal solution
1	100	40	0.4	yes	no	no	no
2	50	35	0.7	no	no	yes	yes
3	45	18	0.4	no	yes	no	yes
4	20	4	0.2	no	yes	yes	no
5	10	10	1.0	no	yes	yes	no
6	5	2	0.4	no	yes	yes	yes
			total weight	100	80	85	100
			total profit	40	34	51	55

- Nu garanteaza gasirea solutiei optime!

Tehnica greedy: Problema rucsacului – varianta fractionara

- Obiectele pot fi *selectate partial* - i.e. putem selecta o *fractiune* din obiect - la o fractiune de profit si greutate



Tehnica greedy: Problema rucsacului – varianta fractionara

Problema rucsacului – varianta fractionara

- Ordonam obiectele in ordine descrescatoare a densitatii de profit (p_i/w_i)
- Procesam urmatorul obiect din lista ordonata, o_i
  - Daca acest obiect incape in totalitate in rucsac il punem in rucsac si continuam la urmatorul obiect
  - Daca obiectul nu incape in totalitate in rucsac atunci punem in rucsac cea mai mare cantitate din obiect si terminam

Complexitatea algoritmului:

- Sortare O(nlogn)
- Procesare O(n)
- Total: O(nlogn)

Aplicatii ale problemei rucsacului – varianta fractionara

- Internet download manager: datele sunt “rupte” in bucati, serverul foloseste acest algoritm si “impacheteaza” bucatile astfel incat sa ocupe toata latimea de banda.
- Vase de transport, camioane de marfa
- Incarcarea bunurilor pe paleti
- Aranjarea coletelor intr-un container
- Planificarea resurselor
- …. Etc

Tehnica greedy: Codurile Huffman

- Tehnica de compresie a datelor – fara pierderi
- Date: secventa de caractere
- Se furnizeaza un ***tabel*** cu frecventa de aparitie a fiecarui caracter
- Pe baza valorilor din tabel se construieste o reprezentare *binara optima, de lungime variabila*
- *Lungimea asteptata = suma frecventelor inmultite cu nr. de biti.*
- *Exemplu*: Avem un fisier cu 100.000 de caractere. Fisierul contine doar literele a,b,c,d,e,f si pt fiecare litera stim frecventa (eg. din tabel apare de 45000 ori, ..)

	a	b	c	d	e	f
Frequency (in thousands)	45	13	12	16	9	5
Fixed-length codeword	000	001	010	011	100	101
Variable-length codeword	0	101	100	111	1101	1100

Tehnica greedy: Codurile Huffman

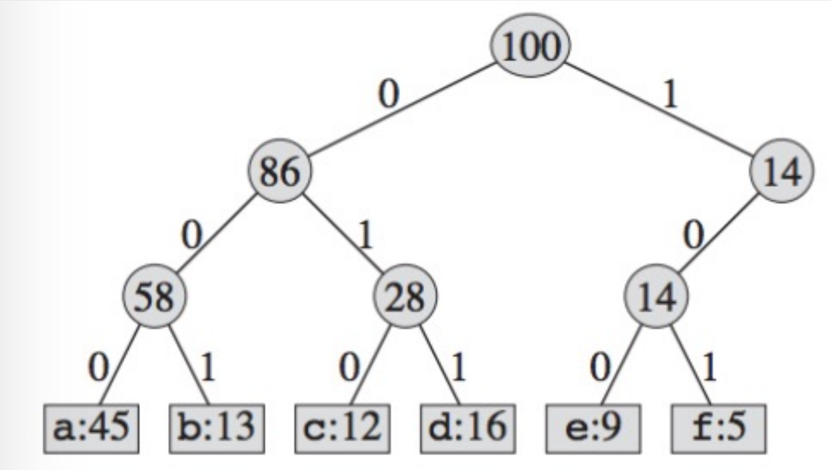
UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

- Fiecarei litere ii asociem un ***cod binar***:
  - de *lungime fixa -> lungime asteptata mesaj:* 300.000 bits
  - de *lungime variabila -> lungime asteptata mesaj:* 224.000 bits (you do the math)
- **Coduri prefix**:
  - consideram doar codurile care nu apar ca si prefix al altui cod
    - metoda optima de compresie
  - simplifica decodificarea: codul care incepe fisierul codat nu este ambiguu; il identificam, il traducem, si continuam pe restul fisierului

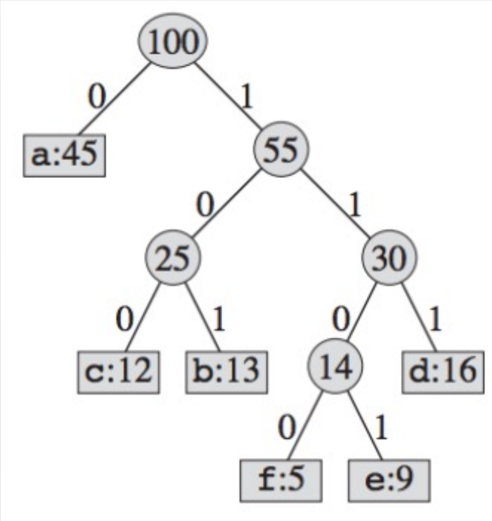
Tehnica greedy: Codurile Huffman

	a	b	c	d	e	f
Frequency (in thousands)	45	13	12	16	9	5
Fixed-length codeword	000	001	010	011	100	101
Variable-length codeword	0	101	100	111	1101	1100

- 001011101 = 0.0.101.1101 = aabe
- Reprezentare: **arbore binar**; codul = calea de la radacina la o frunza; ***0 - "go left"***, ***1 - "go right"***



cod de lungime fixa



cod de lungime variabila

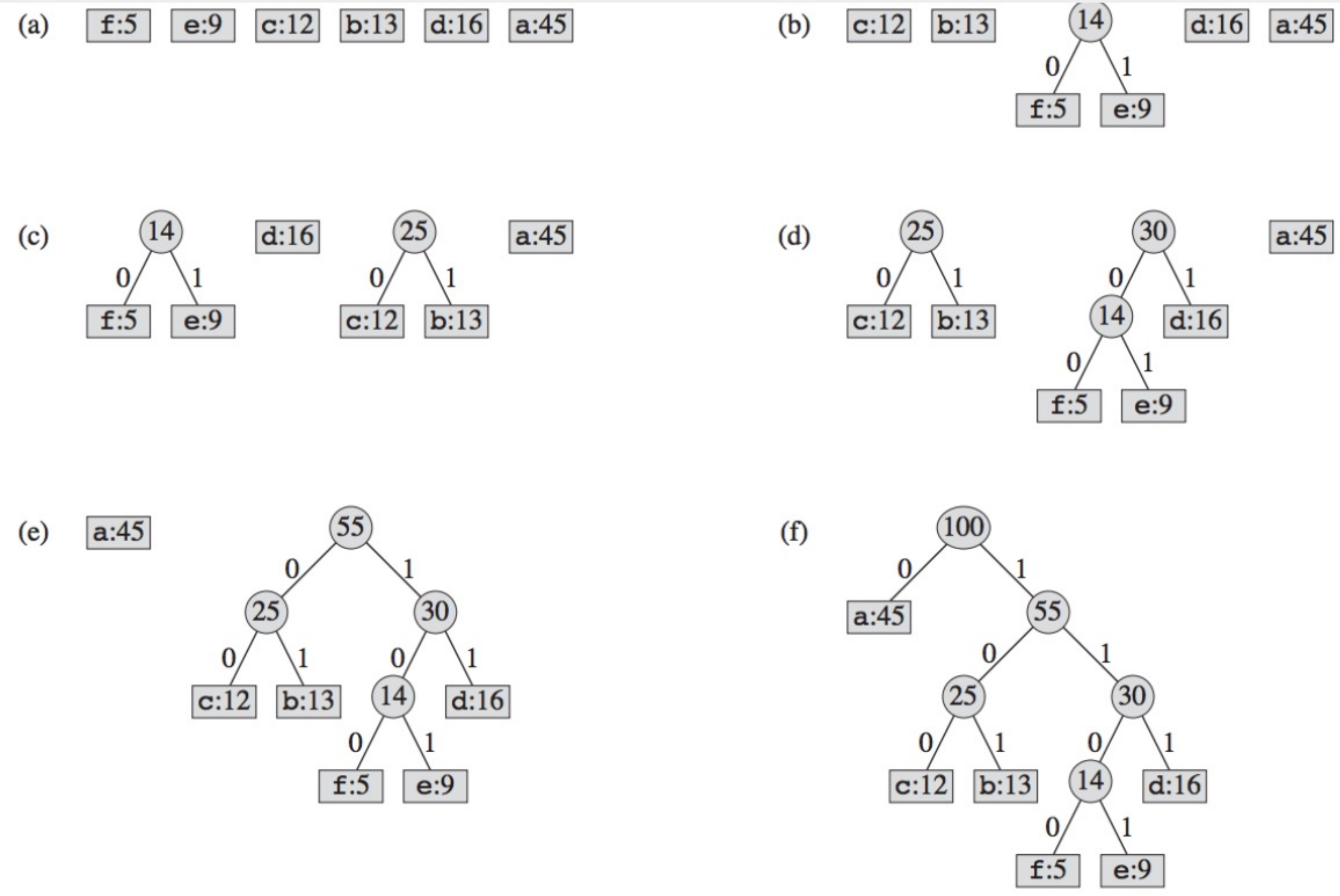
Tehnica greedy: Codurile Huffman

- Cum construim codificarea?
  - C - multimea de *n* caractere
  - strategie bottom-up
  - *Coada de prioritati* (minim) - *Q*, construita pe frecventa, pentru a identifica cele 2 obiecte de frecv. minima pentru a fi unite

```
HUFFMAN(C)
n = |C|
Q = C
for i = 1 to n - 1
    allocate a new node z
    z.left = x = EXTRACT-MIN(Q)
    z.right = y = EXTRACT-MIN(Q)
    z.freq = x.freq + y.freq
    INSERT(Q, z)
return EXTRACT-MIN(Q)    // return the root of the tree
```

Complexitate?

Tehnica greedy: Codurile Huffman



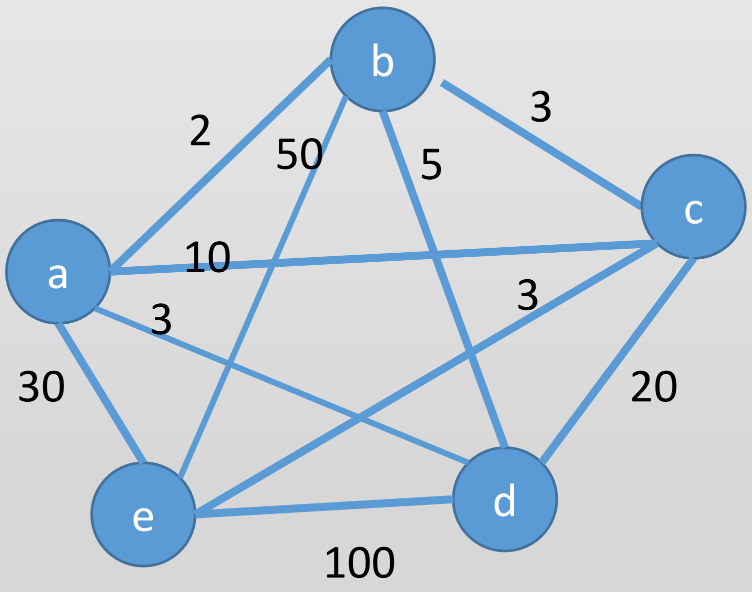
Tehnica greedy: Codurile Huffman aplicatii

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

- In formatele de compresie cum ar fi:
  - GZIP, PKZIP (winzip etc) and BZIP2,
- La formatele de imagini ca JPEG si PNG

Traveling Salesman Problem (TSP)

- **Ciclu Hamiltonian**: Fiind dat un graf un ciclu Hamiltonian este un ciclu simplu care trece prin toate varfurile din graf.
- **TSP**: Fiind dat un graf cu costuri pe muchii, gasiti turul simplu de cost minim.



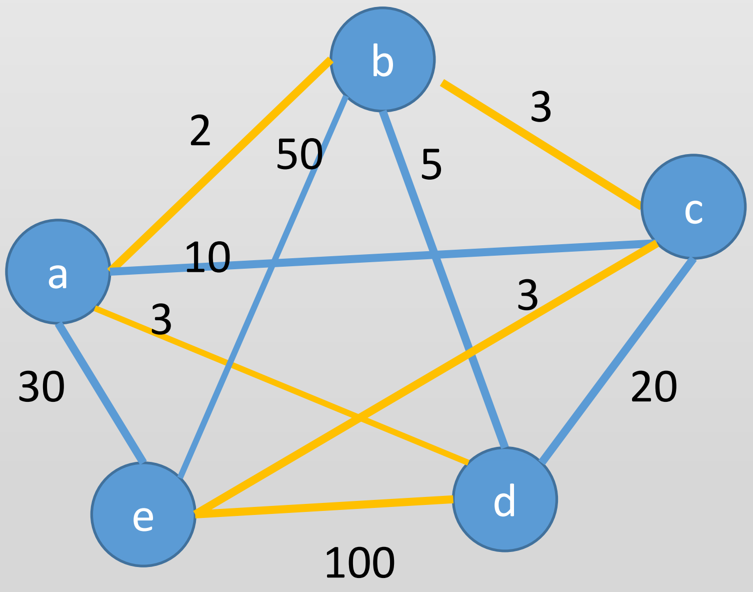
Euristica greedy "Nearest neighbor":

- Alegem orasele (varfurile) pe rand astfel incat de fiecare data alegem orasul cel mai promitator In situatia curenta.
- Daca in situatia curenta au fost alese v_1, v_2, … v_k varfuri, cel mai promitator varf ales urmator va fi varful aflat la distanta minima de v_k.
- Pornim de la varful a:

TSP: Euristica “Nearest neighbor”

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

- **Ciclu Hamiltonian**: Fiind dat un graf un ciclu Hamiltonian este un ciclu simplu care trece prin toate varfurile din graf.
- **TSP**: Fiind dat un graf cu costuri pe muchii, gasiti turul simplu de cost minim.

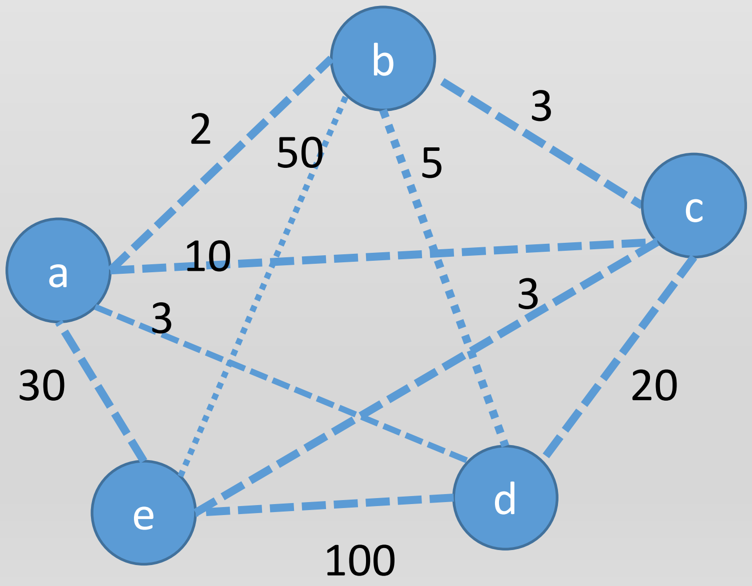


Euristica greedy “Nearest neighbor”:

- Alegem orasele (varfurile) pe rand astfel incat de fiecare data alegem orasul cel mai promitator In situatia curenta.
- Daca in situatia curenta au fost alese v_1, v_2, … v_k varfuri, cel mai promitator varf ales urmator va fi varful aflat la distanta minima de v_k.
- Pornim de la varful a:
  - Strategia greedy genereaza: a, b, c, e, d, a -> cost = 111
  - Solutia optima ar fi: a, d, b, c, e, a -> cost 44

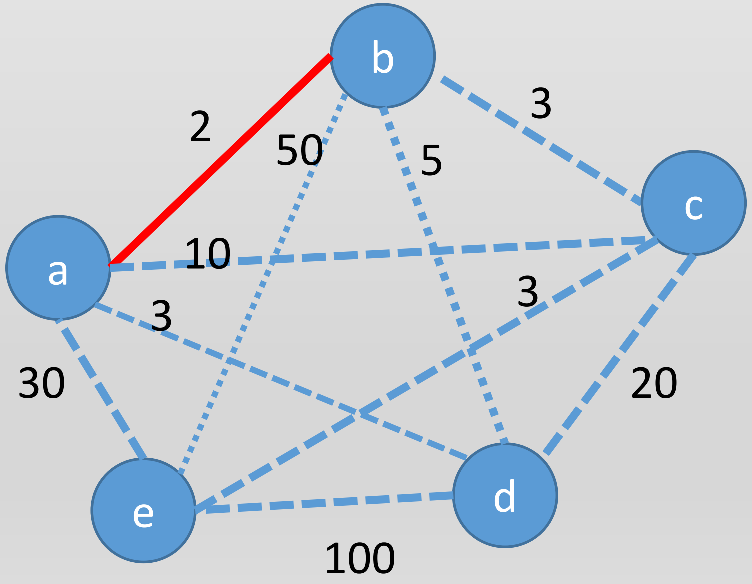
TSP: Euristica “Cost minim muchie”

- Functioneaza pe grafuri complete doar
- Se sorteaza muchiile in ordinea crescatoare a ponderilor
- Pornind de la muchia de cost minim, se selecteaza pe rand cate o muchie noua care lungeste drumul curent, astfel incat:
  - muchia noua sa nu genereze grad >= 3 la nici un nod
  - muchia nu formeaza un ciclu, decat daca am ajuns la ultimul nod din graf



TSP: Euristica “Cost minim muchie”

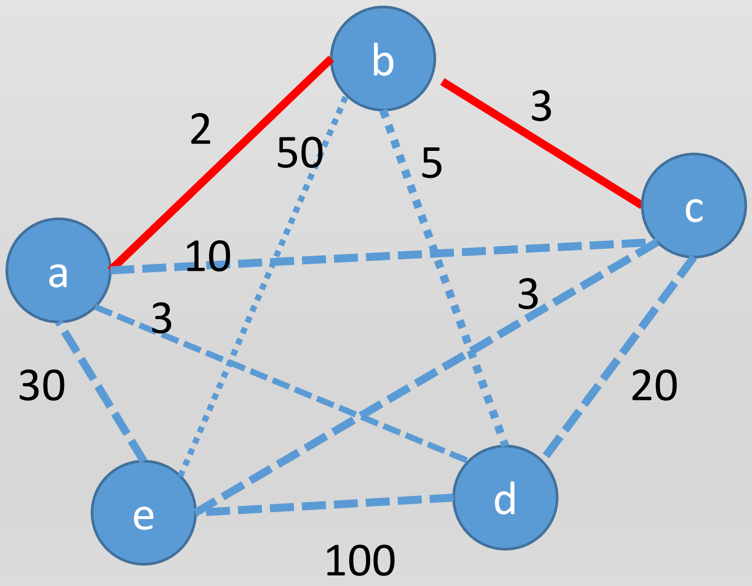
- Functioneaza pe grafuri complete doar
- Se sorteaza muchiile in ordinea crescatoare a ponderilor
- Pornind de la muchia de cost minim, se selecteaza pe rand cate o muchie noua care lungeste drumul curent, astfel incat:
  - muchia noua sa nu genereze grad >= 3 la nici un nod
  - muchia nu formeaza un ciclu, decat daca am ajuns la ultimul nod din graf



a-b = cost 2

TSP: Euristica “Cost minim muchie”

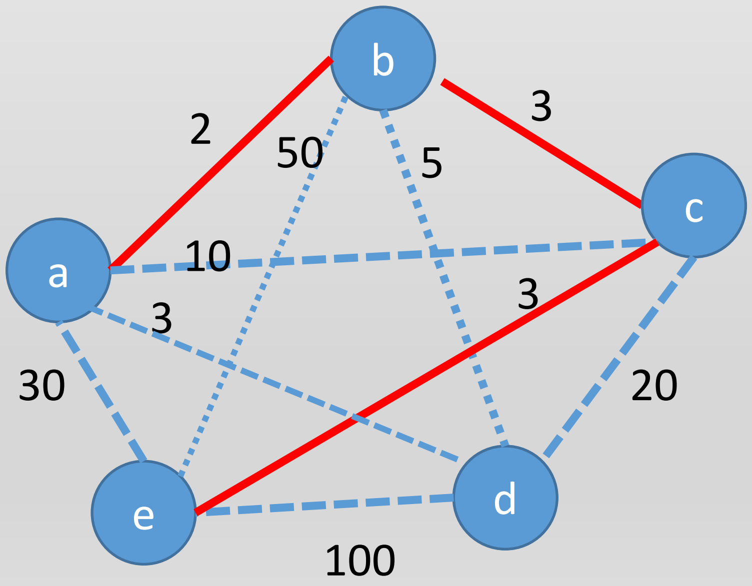
- Functioneaza pe grafuri complete doar
- Se sorteaza muchiile in ordinea crescatoare a ponderilor
- Pornind de la muchia de cost minim, se selecteaza pe rand cate o muchie noua care lungeste drumul curent, astfel incat:
  - muchia noua sa nu genereze grad >= 3 la nici un nod
  - muchia nu formeaza un ciclu, decat daca am ajuns la ultimul nod din graf



a-b = cost 2
b-c cost 3; cost total = 5

TSP: Euristica “Cost minim muchie”

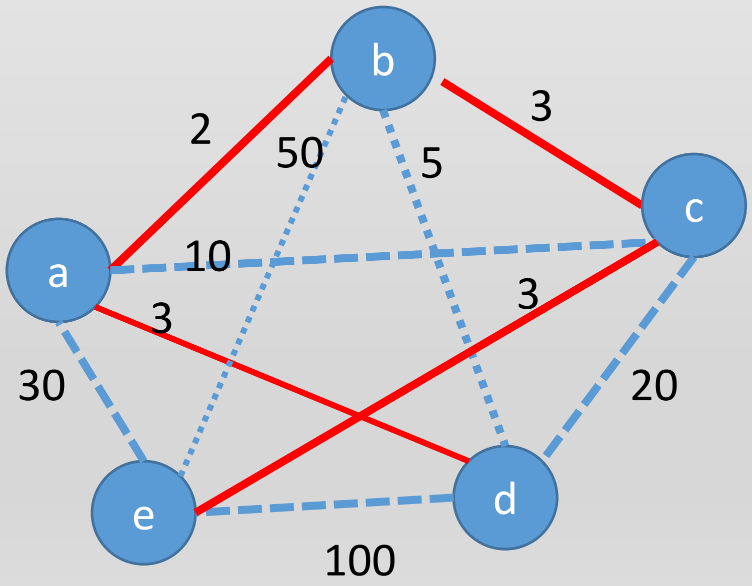
- Functioneaza pe grafuri complete doar
- Se sorteaza muchiile in ordinea crescatoare a ponderilor
- Pornind de la muchia de cost minim, se selecteaza pe rand cate o muchie noua care lungeste drumul curent, astfel incat:
  - muchia noua sa nu genereze grad >= 3 la nici un nod
  - muchia nu formeaza un ciclu, decat daca am ajuns la ultimul nod din graf



a-b = cost 2
b-c cost 3; cost total = 5
c-e cost 3 cost total = 8

TSP: Euristica “Cost minim muchie”

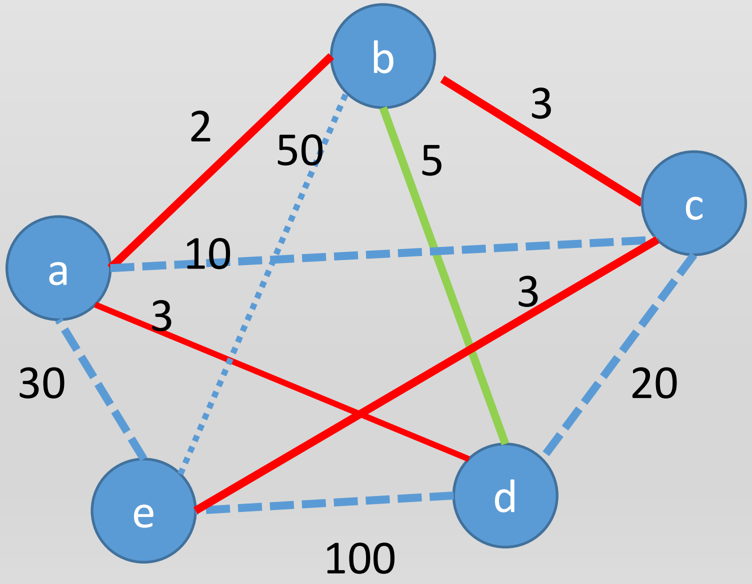
- Functioneaza pe grafuri complete doar
- Se sorteaza muchiile in ordinea crescatoare a ponderilor
- Pornind de la muchia de cost minim, se selecteaza pe rand cate o muchie noua care lungeste drumul curent, astfel incat:
  - muchia noua sa nu genereze grad >= 3 la nici un nod
  - muchia nu formeaza un ciclu, decat daca am ajuns la ultimul nod din graf



a-b = cost 2
b-c cost 3; cost total = 5
c-e cost 3 cost total = 8
a-d cost 3; cost total 11

TSP: Euristica “Cost minim muchie”

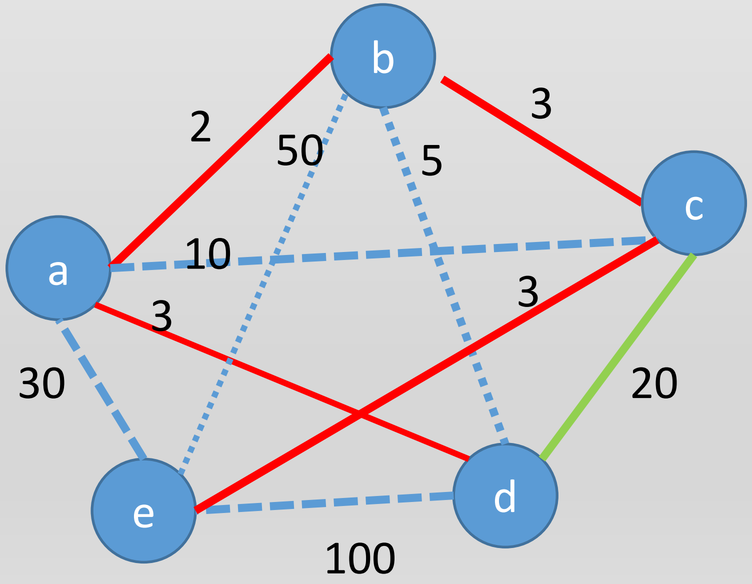
- Functioneaza pe grafuri complete doar
- Se sorteaza muchiile in ordinea crescatoare a ponderilor
- Pornind de la muchia de cost minim, se selecteaza pe rand cate o muchie noua care lungeste drumul curent, astfel incat:
  - muchia noua sa nu genereze grad >= 3 la nici un nod
  - muchia nu formeaza un ciclu, decat daca am ajuns la ultimul nod din graf



a-b = cost 2
b-c cost 3; cost total = 5
c-e cost 3 cost total = 8
a-d cost 3; cost total 11
Putem adauga muchia b-d de cost 5 ?

TSP: Euristica “Cost minim muchie”

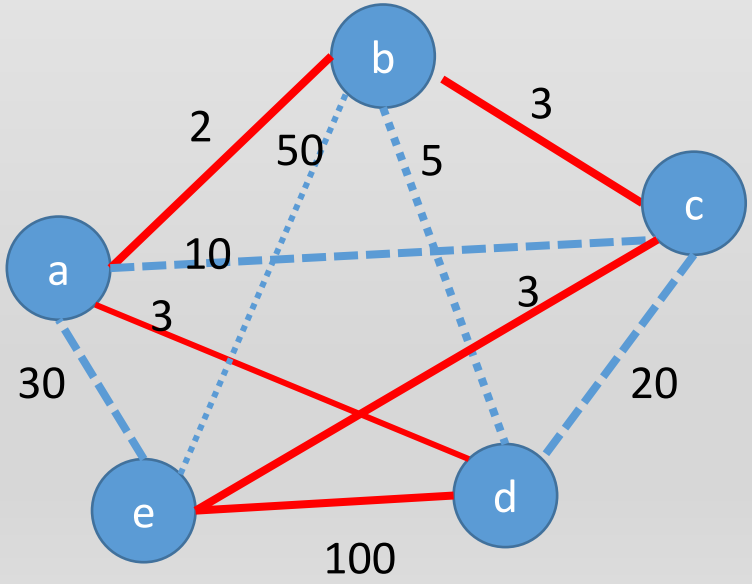
- Functioneaza pe grafuri complete doar
- Se sorteaza muchiile in ordinea crescatoare a ponderilor
- Pornind de la muchia de cost minim, se selecteaza pe rand cate o muchie noua care lungeste drumul curent, astfel incat:
  - muchia noua sa nu genereze grad >= 3 la nici un nod
  - muchia nu formeaza un ciclu, decat daca am ajuns la ultimul nod din graf



a-b = cost 2
b-c cost 3; cost total = 5
c-e cost 3 cost total = 8
a-d cost 3; cost total 11
Dar muchia c-d?

TSP: Euristica “Cost minim muchie”

- Functioneaza pe grafuri complete doar
- Se sorteaza muchiile in ordinea crescatoare a ponderilor
- Pornind de la muchia de cost minim, se selecteaza pe rand cate o muchie noua care lungeste drumul curent, astfel incat:
  - muchia noua sa nu genereze grad >= 3 la nici un nod
  - muchia nu formeaza un ciclu, decat daca am ajuns la ultimul nod din graf



a-b = cost 2
b-c cost 3; cost total = 5
c-e cost 3 cost total = 8
a-d cost 3; cost total 11
e-d cost 100 – singura care mai poate fi adaugata:
cost total: 111

Greedy vs Backtracking in probleme de optimizare combinatoriala

Greedy	Backtracking
- Problema se descompune in **componente**	- Problema se descompune in **componente**
- La fiecare pas se decide valoarea unei componente a solutiei	- La fiecare pas se decide valoarea unei componente a solutiei
- Alegere bazata pe optim local	- Alegere neinformata
- Deciziile sunt finale	- Se poate reveni asupra deciziei
- Se dezvolta o singura solutie	- Se dezvolta toate solutiile fezabile (*)

(*) – la Branch and Bound se mai elimina si solutiile partiale care nu pot conduce la o solutie globala mai buna decat cea mai buna solutie gasita pana la acel moment

Greedy vs Programare Dinamica in probleme de optimizare combinatoriala

Greedy	DP
- Problema se descompune in **componente**	- Problema se descompune in **sub-probleme**
- La fiecare pas se decide valoarea unei componente a solutiei	- La fiecare pas se gaseste solutia optima la o sub-problema
- Alegere bazata pe optim local	- Alegere bazata pe solutia optima la sub-probleme; garantia optim global (proprietatea de sub-structura optima)
- Deciziile sunt finale	- Deciziile sunt finale
- Se dezvolta o singura solutie	- Se dezvolta cate o solutie optima la sub-probleme

Exercitii

Fiind date urmatoarele probleme sa se rezolve atat folosind greedy cat si folosind backtracking. Exemplificati rularea pas cu pas si discutati rezultatele intermediare si eficienta celor doua implementari.

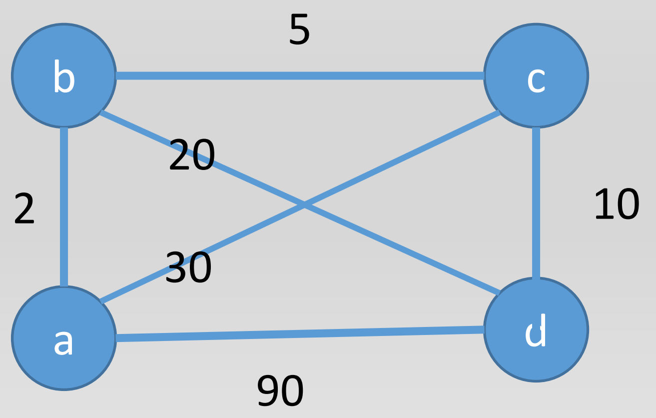
1. Problema returnarii restului

   Se dau urmatoarele valor de bancnote:

   a) D = {1; 2; 5; 10; 20; 20; 25}, suma S = 40.

   b) D = {1; 1; 1; 5; 8; 10}, Suma S = 13

2. Problema comis-voiajorului (Traveling Salesman Problem) pentru graful din figura pornind de la varful a:



Referinte

1. Th. Cormen et al.: Introduction to Algorithms, cap. 16 [mandatory]

Optional:

1. S. Skiena: The Algorithm Design Manual, cap 7
2. https://www.cs.princeton.edu/~wayne/kleinberg-tardos/
3. https://ro.wikipedia.org/wiki/Leu_rom%C3%A2nesc
4. https://en.wikipedia.org/wiki/Knapsack_problem
5. https://en.wikipedia.org/wiki/Greedy_algorithm
6. https://web.stanford.edu/class/archive/cs/cs161/cs161.1138/
7. http://www.radford.edu/~nokie/classes/360/greedy.html
8. https://www.cs.rochester.edu/~gildea/csc282/slides/C16-greedy.pdf